SHARP

SHARP
COMPUTER
SOFTWARE

X68000用

C COMPILER PRO-68K ver2.0

Cライブラリマニュアル VOL.2

X68000用

C COMPILER PRO-68K ver2.0

Cライブラリマニュアル VOL.2

SHARP

はじめに

本書は、「C compiler PRO-68K ver2.0」（以下、XCコンパイラと表記します）でプログラム開発を行うときに必要となるCライブラリの概要と使用方法、大文字関数リファレンスに関して記述しています。
小文字関数のリファレンスに関しては、「Cライブラリマニュアル VOL.1」をご覧ください。

なお、別冊の「Cユーザーズマニュアル」にXCコンパイラの商品構成、動作環境の作成方法、各マニュアルの内容が説明されています。
初めて本プログラムをご使用になる方は、必ず「Cユーザースマニュアル」を先にご覧ください。

本書は、C言語について一定以上の知識を有し、プログラミング経験のあるユーザーが、Cライブラリの機能と使用方法を理解してX68000の豊富な機能をフルに活用できるようになること、さらに進んでユーザーが自分自身でもライブラリを作成できるようになることを目的として書かれています。

なお、C言語の書式・文法については「Cリファレンスマニュアル」、開発環境の設定や開発手順などについては「Cユーザーズマニュアル」を参照してください。

Cライブラリの概要

Human68k の C ランタイムライブラリは、C 言語でプログラム開発を行う際に必要となる関数、およびマクロで構成されています。
C 言語では、データの入出力などハードウェアに依存する処理は、言語仕様からはずし、これを関数として提供しています。
また、文字列操作、およびサーチやソートといった、使用頻度の高い汎用的な処理についても関数として提供し、C プログラマの労力を軽減しています。
Human68k の C ランタイムライブラリでは、上記のような C 標準ライブラリの他に、日本語処理をサポートする関数や、IOCS コールライブラリ、DOS コールライブラリ、BASIC ライブラリといった、Human68k 固有の関数が豊富に用意され、その処理能力はきわめて強力なものとなっています。

たとえば、IOCS コールライブラリによって、キーボードやプリンタといった標準的な入出力機器だけでなく、ジョイスティック、RS-232C、ADPCM、FM 音源、マウス、グラフィック、スプライト機能といった、多様な入出力操作が可能となります。
また、DOS コールライブラリによって、日本語処理、日付・時刻の制御、環境変数の操作などの、Human68k オペレーティングシステムの機能を有効に利用することができます。
BASIC ライブラリは、BASTOC において使用し、BASIC 言語から C 言語への変換を容易にしています。
また、これらの関数をサポートするために、専用のインクルードファイルが用意されています。

C ランタイムライブラリの項目別の分類、およびその項目に属する関数の名前と機能については、「VOL.1　第 3 章　項目別ランタイムルーチン」を参照してください。

本書の構成

本書は、以下の内容から構成されています。

第1章　リファレンス（大文字）

各ライブラリのリファレンスであり、実際にライブラリを使用するときに使用の手引きとして利用してください。
この章では、XC コンパイラがサポートするライブラリ関数（大文字）を、リファレンス形式でアルファベット順に掲載しています。
関数には、レベルが設定されています。
これは、C 言語で一般的にいわれている入出力のレベルとは違い、XC コンパイラ独自のものです。

付録

ユーザーライブラリの作成方法を、具体的な例にそって説明します。

索引

50 音順、アルファベット順で整理しています。

関数索引

大文字関数、および小文字関数をアルファベット順で整理しています。

機能別関数索引

関数を分かりやすく機能別に分類しています。

CONTENTS

第１章

リファレンス
（大文字）

この章では、XC コンパイラがサポートするライブラリ関数（大文字）を、リファレンス形式でアルファベット順に掲載しています。
関数には、レベルが設定されています。
これは、C 言語で一般的にいわれている入出力のレベルとは違い、XC コンパイラ独自のものです。
レベルが表す意味は、次の通りです。

- レベル 3：BASIC 関連の関数（ライブラリファイル BASLIB. L に含まれる）のうち、他のレベルの関数と混在させて使うときには、注意しなければならないもの。
- レベル 2：ライブラリファイル CLIB. L に含まれるすべての関数、および、これらの関数と混在させて使用できるもの。
- レベル 1：DOS コール関連の関数（ライブラリファイル DOSLIB. L に含まれる）のうち、他のレベルの関数と混在させて使うときには、注意しなければならないもの。
- レベル 0：IOCS コール関連の関数（ライブラリファイル IOCSLIB. L に含まれる）のうち、他のレベルの関数と混在させて使うときには、注意しなければならないもの。

レベル 2 以外の関数を使用するときは、なるべく同一レベルの関数だけでプログラムを記述してください。

ABORTJOB

レベル0

書　式

```
#include <iocslib.h>

void ABORTJOB ( ) ;
```

機　能

アポート処理を行います。
本ファンクションコールはHuman68kが使用します。

戻り値

戻り値はありません。
リターンせずに、エラーアボートルーチン（OSのエラーが出たとき、中止〈A〉を選ぶ）へ制御を移行します。

ABORTRST

レベル 0

書式	# include <iocslib. h> void ABORTRST ( ) ;
機能	アボートするためのフラグを、リセットします。 本ファンクションコールは Human68k が使用します。
戻り値	戻り値はありません。

ADPCMAIN

レベル 0

書　式

```
#include <iocslib.h>

void ADPCMAIN (TBLADDRESS, MODE, TBLCNT) ;
struct CHAIN  *TBLADDRESS ; /*  チェーンテーブルのアドレス  */
int MODE ; /*  モード  */
int TBLCNT ; /*  チェーンテーブルの個数  */
struct CHAIN {
  int adr ; /*  先頭アドレス  */
  unsigned short len ; /*長さ  */
} ;
```

機　能

ADPCMからデータを入力します。
データ長は0x0001～0xFFFFです。
TBLADDRESSには入力データチェーンテーブルのアドレスを指定します。
データチェーンテーブルの内容は、先頭アドレス，長さ……先頭アドレス，長さの順番で定義しておきます。
MODEには、サンプリング周波数×256＋出力モードを指定します。

●サンプリング周波数

0 ：　3.9KHz（1950バイト/sec）
1 ：　5.2KHz（2600バイト/sec）
2 ：　7.8KHz（3900バイト/sec）
3 ：10.4KHz（5200バイト/sec）
4 ：15.6KHz（7800バイト/sec）

●出力モード

0 ：　音声出力カット
1 ：　音声出力左
2 ：　音声出力右
3 ：　音声出力両方

TBLCNTには入力データチェーンテーブルの個数を指定します。

データ入力はモノラルで入力しますが、モニタ出力を出力モードで指定できます。

戻り値

戻り値はありません。

プログラム例

```
#include        <stdlib.h>
#include        <iocslib.h>

#define SFREQ   1               /* サンプリング周波数(5.2KHz) */
#define SOUND   2               /* 入力モード(音声入力右) */
#define TSIZE   3               /* テーブルサイズ */
#define BSIZE   4096

static struct CHAIN     table[TSIZE];   /* chainテーブル */


main()
{
        int     set_table(struct CHAIN *, int);
        int     mode;

        if (set_table(table, TSIZE) == TSIZE) {
                mode = SFREQ * 256 + SOUND;
                ADPCMAIN(table, mode, TSIZE);
        }
}


static int      set_table(struct CHAIN *p, int n)
{
        int     i;

        for (i = 0; i < n; i++) {
                /* The member 'adr' is badly declared */
                if ((p->adr = (INT)malloc(BSIZE)) == NULL)
                        break;
                p->len = BSIZE;
                ++p;
        }
        return (i);
}
```

ADPCMAOT

レベル 0

書　式

```
#include <iocslib. h>

void ADPCMAOT (TBLADDERSS, MODE, TBLCNT) ;
struct CHAIN ＊TBLADDRESS ; /＊ チェーンテーブルのアドレス ＊/
int MODE; /＊ モード ＊/
int TBLCNT; /＊ チェーンテーブルの個数 ＊/
struct CHAIN {
  int adr; /＊ 先頭アドレス ＊/
  unsigned short; /＊ 長さ ＊/
};
```

機　能

ADPCM へデータを出力します。
データ長は、0x0001～0xFFFF です。
TBLADDRESS には、出力データチェーンテーブルのアドレスを指定します。
アドレスの内容は、先頭アドレス，長さ……先頭アドレス，長さの順番で定義しておきます。
MODE には、サンプリング周波数×256＋出力モードを指定します。

●サンプリング周波数

0 ： 3.9KHz (1950 バイト/sec)
1 ： 5.2KHz (2600 バイト/sec)
2 ： 7.8KHz (3900 バイト/sec)
3 ： 10.4KHz (5200 バイト/sec)
4 ： 15.6KHz (7800 バイト/sec)

●出力モード

0 ： 音声出力カット
1 ： 音声出力左
2 ： 音声出力右
3 ： 音声出力両方

TBLCNT には、出力データチェーンテーブルの個数を指定します。

戻り値

戻り値はありません。

プログラム例

```
#include        <stdlib.h>
#include        <iocslib.h>

#define SFREQ   1               /* サンプリング周波数(5.2KHz) */
#define SOUND   1               /* 出力モード(音声出力左) */
#define TSIZE   3               /* テーブルサイズ */
#define BSIZE   4096

static struct CHAIN     table[TSIZE];   /* chainテーブル */


main()
{
        int     set_table(struct CHAIN *, int);
        int     mode;

        if (set_table(table, TSIZE) == TSIZE) {
                mode = SFREQ * 256 + SOUND;
                ADPCMAIN(table, mode, TSIZE);
                ADPCMAOT(table, mode, TSIZE);
        }
}


static int      set_table(struct CHAIN *p, int n)
{
        int     i;

        for (i = 0; i < n; i++) {
                /* The member 'adr' is badly declared */
                if ((p->adr = (INT)malloc(BSIZE)) == NULL)
                        break;
                p->len = BSIZE;
                ++p;
        }
        return (i);
}
```

ADPCMINP

レベル0

書　式

```
#include <iocslib.h>

void ADPCMINP (ADDRESS, MODE, LENGTH) ;
unsigned char *ADDRESS; /* データ読み込み領域のアドレス */
int MODE; /* モード */
int LENGTH; /* 入力データ長 */
```

機　能

ADPCMからデータを入力します。

データ長が0xFF00以上の場合、入力処理は0xFF00単位で行われるので、リターンまでに時間がかかります。

ADDRESSには、データ読み込み領域のアドレスを指定します。

MODEには、サンプリング周波数×256＋出力モードを指定します。

●サンプリング周波数

0 ： 3.9KHz (1950バイト/sec)
1 ： 5.2KHz (2600バイト/sec)
2 ： 7.8KHz (3900バイト/sec)
3 ： 10.4KHz (5200バイト/sec)
4 ： 15.6KHz (7800バイト/sec)

●出力モード

0 ： 音声出力カット
1 ： 音声出力左
2 ： 音声出力右
3 ： 音声出力両方

LENGTHには、入力データの長さを指定します。

データ入力はモノラルで入力しますが、モニタ出力を出力モードで指定できます。

戻り値

戻り値はありません。

プログラム例

```
#include        <iocslib.h>

#define SFREQ   1               /* サンプリング周波数(5.2KHz) */
#define SOUND   3               /* 入力モード(音声入力両方) */
#define DTLEN   8192            /* データ長 */

static char     dbuf[DTLEN];            /* データ読込み領域 */

main()
{
        int     mode;

        mode = SFREQ * 256 + SOUND;
        ADPCMINP(dbuf, mode, DTLEN);
}
```

ADPCMLIN

レベル 0

書　式

```
#include <iocslib.h>

void ADPCMLIN (TBLADDRESS, MODE);
struct CHAIN2 * TBLADDRESS; /*  アレイチェーンテーブルのアドレス
*/
int MODE; /*  モード  */
struct CHAIN2 {
  int adr; /*  先頭アドレス  */
  unsigned short len; /*  長さ  */
  struct CHAIN 2  *next; /*次のテーブルのアドレス  */
};
```

機　能

ADPCM からデータを入力します。
TBLADDRESS には、入力データアレイチェーンテーブルの先頭アドレスを指定します。
TBLADDRESS が指す領域には先頭アドレス，長さ，次のテーブルアドレスが格納され、次のテーブルにチェーンしています。
最後のテーブルでは、次のテーブルアドレスには 0 を設定します。
MODE には、サンプリング周波数×256＋出力モードを指定します。

●サンプリング周波数

0 ： 3.9KHz (1950 バイト/sec)
1 ： 5.2KHz (2600 バイト/sec)
2 ： 7.8KHz (3900 バイト/sec)
3 ： 10.4KHz (5200 バイト/sec)
4 ： 15.6KHz (7800 バイト/sec)

●出力モード

0 ： 音声カット
1 ： 音声出力左
2 ： 音声出力右
3 ： 音声出力両方

データ入力はモノラルで入力しますが、モニタ出力を出力モードで指定できます。

戻り値

戻り値はありません。

プログラム例

```
#include        <stdlib.h>
#include        <iocslib.h>

#define SFREQ   0               /* サンプリング周波数(3.9KHz) */
#define SOUND   3               /* 出力モード(音声出力両方) */
#define TSIZE   3
#define BSIZE   4096

typedef struct CHAIN2   CH2;


main()
{
        CH2     *set_chain(int);
        CH2     *chain;         /* chainテーブル */
        int     mode;           /* モード */

        if ((chain = set_chain(TSIZE)) != NULL) {
                mode = SFREQ * 256 + SOUND;
                ADPCMLIN(chain, mode);
        }
}


static CH2      *set_chain(int n)
{
        CH2     *ip;
        CH2     *cp;
        CH2     *pp;
        int     i;

        for (i = 0; i < n; i++) {
                if ((cp = (CH2 *)malloc(sizeof(CH2))) == NULL)
                        return (NULL);
                cp->len = BSIZE;
                /* The member 'adr' is badly declared */
                if ((cp->adr = (INT)malloc(BSIZE)) == NULL)
                        return (NULL);
                if (i == 0)
                        ip = cp;
                if (i >= 0)
                        pp->next = cp;
                pp = cp;
                ++cp;
        }
        pp->next = NULL;
        return (ip);
}
```

ADPCMLOT

レベル0

書　　式

```
#include <iocslib.h>

void ADPCMLOT (TBLADDRESS, MODE) ;
struct CHAIN2 * TBLADDRESS ; /*  アレイチェーンテーブルのアドレス
*/
int MODE ;  /*  モード  */
struct CHAIN2 {
  int adr ;  /*  先頭アドレス  */
  unsigned short len ;  /*  長さ  */
  struct CHAIN2  *next ;  /*次のテーブルのアドレス  */
} ;
```

機　　能

ADPCMへデータを出力します。
TBLADDRESSには、出力データアレイチェーンテーブルの先頭アドレスを指定します。
TBLADDRESSが指す領域には、先頭アドレス，長さ，次のテーブルアドレスが格納され、次のテーブルにチェーンしています。
最後のテーブルでは、次のテーブルアドレスには0を設定します。
MODEには、サンプリング周波数×256＋出力モードを指定します。

●サンプリング周波数

0 ： 3.9KHz (1950バイト/sec)
1 ： 5.2KHz (2600バイト/sec)
2 ： 7.8KHz (3900バイト/sec)
3 ： 10.4KHz (5200バイト/sec)
4 ： 15.6KHz (7800バイト/sec)

●出力モード

0 ： 音声出力カット
1 ： 音声出力左
2 ： 音声出力右
3 ： 音声出力両方

戻 り 値

戻り値はありません。

プログラム例

```
#include        <stdlib.h>
#include        <iocslib.h>

#define SOUND   3               /* 出力モード(音声出力両方) */
#define SFREQ   2               /* サンプリング周波数(7.8KHz) */
#define TSIZE   3
#define BSIZE   4096

typedef struct CHAIN2   CH2;


main()
{
        CH2     *set_chain(int);
        CH2     *chain;         /* chainテーブル */
        int     mode;           /* モード */

        if ((chain = set_chain(TSIZE)) != NULL) {
                mode = SFREQ * 256 + SOUND;
                ADPCMLIN(chain, mode);
                ADPCMLOT(chain, mode);
        }
}


static CH2      *set_chain(int n)
{
        CH2     *ip;
        CH2     *cp;
        CH2     *pp;
        int     i;

        for (i = 0; i < n; i++) {
                if ((cp = (CH2 *)malloc(sizeof(CH2))) == NULL)
                        return (NULL);
                cp->len = BSIZE;
                /* The member 'adr' is badly declared */
                if ((cp->adr = (INT)malloc(BSIZE)) == NULL)
                        return (NULL);
                if (i == 0)
                        ip = cp;
                if (i >= 0)
                        pp->next = cp;
                pp = cp;
                ++cp;
        }
        pp->next = NULL;
        return (ip);
}
```

ADPCMMOD

レベル0

書　式　# include <iocslib. h>

void ADPCMMOD (MODE) ;
int MODE; /* モード */

機　能　ADPCMの実行制御を行います。
MODEには、次の動作モードを指定します。

0 ： 終了
1 ： 中止
2 ： 再開

戻り値　戻り値はありません。

プログラム例

```
#include        <iocslib.h>

#define SFREQ   1               /* サンプリング周波数(5.2KHz) */
#define SOUND   3               /* 出力モード(音声出力両方) */
#define DTLEN   4096            /* データ長 */
#define EMODE   2               /* 実行モード(実行再開) */

static char     dbuf[DTLEN];    /* 出力データ */

main()
{
        int     mode;

        ADPCMMOD(EMODE);

        mode = SFREQ * 256 + SOUND;
        ADPCMOUT(dbuf, mode, DTLEN);
}
```

ADPCMOUT

レベル 0

書　式

```
#include <iocslib.h>

void ADPCMOUT (ADDRESS, MODE, LENGTH) ;
unsigned char *ADDRESS; /* 出力データアドレス */
int MODE; /* モード */
int LENGTH; /* 出力データ長 */
```

機　能

ADPCM へデータを出力します。
データ長が 0xFF00 以上の場合、出力処理は 0xFF00 単位で行われるので、リターンまでに時間がかかります。
ADDRESS には、出力データのアドレスを指定します。
MODE には、サンプリング周波数×256＋出力モードを指定します。

●サンプリング周波数

0 ： 3.9KHz (1950 バイト/sec)
1 ： 5.2KHz (2600 バイト/sec)
2 ： 7.8KHz (3900 バイト/sec)
3 ： 10.4KHz (5200 バイト/sec)
4 ： 15.6KHz (7800 バイト/sec)

●出力モード

0 ： 音声出力カット
1 ： 音声出力左
2 ： 音声出力右
3 ： 音声出力両方

LENGTH には、出力データ長を指定します。

戻り値

戻り値はありません。

プログラム例

```
#include        <iocslib.h>

#define SFREQ   1               /* サンプリング周波数(5.2KHz) */
#define SOUND   3               /* 出力モード(音声出力両方) */
#define DTLEN   8192            /* データ長 */

static char     dbuf[DTLEN];    /* 出力データ */

main()
{
        int     mode;

        mode = SFREQ * 256 + SOUND;
        ADPCMOUT(dbuf, mode, DTLEN);
}
```

ADPCMSNS

レベル 0

書式 # include <iocslib. h>

int ADPCMSNS ( );

機能 ADPCM の実行モードを調べます。

戻り値 ADPCM の実行モードを返します。

0x00 ： 何もしていない
0x02 ： 出力中 (ADPCMOUT 実行中)
0x04 ： 入力中 (ADPCMINP 実行中)
0x12 ： 出力中 (ADPCMAOT 実行中)
0x14 ： 入力中 (ADPCMAIN 実行中)
0x22 ： 出力中 (ADPCMLOT 実行中)
0x24 ： 入力中 (ADPCMLIN 実行中)

プログラム例

```
#include        <iocslib.h>

#define SFREQ   1               /* サンプリング周波数(5.2KHz) */
#define SOUND   3               /* 出力モード(音声出力両方) */
#define DTLEN   4096            /* データ長 */

static char     dbuf[DTLEN];    /* 出力データ */

main()
{
        int     mode;
        int     result;

        result = ADPCMSNS();
        printf("sns = %d¥n", result);
        if (result == 0) {
                mode = SFREQ * 256 + SOUND;
                ADPCMOUT(dbuf, mode, DTLEN);
        }
}
```

AKCONV

レベル 0

書　式

```
#include <iocslib.h>

int AKCONV (MODE, CODE) ;
int CODE; /*  ANKコード  */
int MODE; /*  文字種  */
```

機　能

ANKコードをシフトJISコードに変換します。
CODEにはANKコードを指定します（0x20～0x7E、0xA1～0xDF)。
MODEには文字種を指定します。

0：ひらがな
1：カタカナ

戻り値

シフトJISコードを返します。
ただし、上位16ビットが0xFFFFならエラーが発生したことを示します。

ALARMGET

レベル 0

書　　式

#include <iocslib. h>

int ALARMGET (DATETIME, OFFTIME, JOB) ;
int ＊DATETIME ; /＊ アラームの時間を指すポインタ ＊/
int ＊OFFTIME ; /＊ オフされるまでの時間を指すポインタ ＊/
int ＊JOB ; /＊ 処理アドレス ＊/

機　　能

アラームの時間と処理アドレスを読み込みます。

戻 り 値

処理アドレス (ALARMSET 参照) が返されます。
そして、アラームの時間と処理アドレスが、DATETIME, OFFTIME, JOB の各変数に格納されます。
DATETIME の形式と設定内容は次の通りです。
日・時・分はいずれも BCD2 桁です。

0000WWWW DDDDDDDD HHHHHHHH MMMMMMMM

WWWW	：	曜日 (0～6)	0 ： 日曜日
DDDDDDDD	：	日 (01～31)	
HHHHHHHH	：	時 (00～23)	
MMMMMMMM	：	分 (00～59)	

曜日が 0xF、日，時，分が 0xFF になっている場合は、無指定を表します。

OFFTIME には、テレビあるいはコンピュータが OFF されるまでの時間が設定されます(単位は分)。
JOB には処理アドレスが設定されます (ALARMSET 参照)。

プログラム例

```
#include        <stdio.h>
#include        <iocslib.h>

main()
{
        int     time;           /* 日付・時刻 */
        int     offtime;        /* offになるまでの時間 */
        int     job;            /* 処理アドレス */

        ALARMMOD(1);            /* アラーム許可状態にする */

        ALARMGET(&time, &offtime, &job);

        printf("     job = 0x%x¥n", job);
        printf("    曜日 = %x¥n", (time >> 24) & 0x0f);
        printf("      日 = %x¥n", (time >> 16) & 0xff);
        printf("      時 = %x¥n", (time >> 8) & 0xff);
        printf("      分 = %x¥n", (time >> 0) & 0xff);
        printf("長さ(分) = %x¥n", offtime);
}
```

ALARMMOD

レベル0

書　　式　#include <iocslib.h>

int ALARMMOD (MODE) ;
int MODE ; /* モード */

機　　能　アラームの禁止/許可の設定、および現在の状態の通知を行います。
MODEには次の内容を指定できます。

0 ： 禁止
1 ： 許可
2 ： 通知要求

戻り値　設定されたアラームの状態を返します。

0 ： 禁止
1 ： 許可

プログラム例

```
#include        <stdio.h>
#include        <stdlib.h>
#include        <iocslib.h>

#define M_UNABL         0       /* 禁止*/
#define M_ENABL         1       /* 許可 */
#define M_SENSE         2       /* 通知要求 */

main()
{
        if (ALARMMOD(M_SENSE) == M_UNABL) {
                printf("アラームは禁止されています¥n");
                exit(1);
        }
}
```

ALARMSET

レベル 0

書　式

```
# include <iocslib. h>

int ALARMSET (DATETIME, OFFTIME, MODE) ;
int DATETIME;  /*  日付・時間  */
int OFFTIME;  /*  オフまでの時間  */
int MODE;  /*  モード  */
```

機　能

アラームの時間と処理アドレスを設定します。
DATETIME の形式と指定内容は次の通りです。
日・時・分はすべて BCD2 桁です。

0000WWWW DDDDDDDD HHHHHHHH MMMMMMMM

WWWW	： 曜日（0〜6）	0 ：	日曜日
DDDDDDDD	： 日（01〜31）		
HHHHHHHH	： 時（00〜23）		
MMMMMMMM	： 分（00〜59）		

曜日に 0xF、日，時，分に 0xFF を指定すると、それぞれが無指定になります。
ただし、すべてを無指定にしないでください。

OFFTIME には、テレビまたはコンピュータが OFF されるまでの時間を設定します。

0 ： いつまでも OFF にしない
?? ： OFF するまでの時間を 1 分単位で指定する

MODE には処理アドレスを指定します。

0 ： パワー ON して、テレビオン&コンピュータモードにする。
−1 ： パワー ON のみ。テレビコントロールは行いません。
1〜0x3F ： テレビコントロールを行います。（TVCTRL 参照）
0x40〜0xFFFFFFFE： 処理アドレス。

処理アドレスのプログラムの先頭は、0x60 でなければなりません。

戻り値　エラーのとき −1 を返します。

プログラム例

```
#include        <iocslib.h>

#define YOBI    0x00            /* 曜日 (BCD) */
#define DATE    0x25            /* 日 (BCD) */
#define TIME    0x12            /* 時 (BCD) */
#define MINT    0x00            /* 分 (BCD) */

main()
{
        int     dt;             /* 日付・時刻 */
        int     length;         /* offになるまでの時間 */
        int     mode;           /* モード */
        int     almmode;        /* アラームモード */


        length = 10;
        mode = 0;
        almmode = 1;

        ALARMMOD(almmode);      /* アラーム許可 */

        dt = YOBI << 24;
        dt |= DATE << 16;
        dt |= TIME << 8;
        dt |= MINT << 0;

        ALARMSET(dt, length, mode);     /* アラーム設定 */
}
```

ALLCLOSE

レベル 1

書　式　# include <doslib. h>

```
void ALLCLOSE ( ) ;
```

機　能　現在オープンしているファイルハンドルを、すべてクローズします。
子プロセスがオープンしたファイルハンドルについても同様です。
ただし、該当プロセスの親プロセスがオープンしたファイルハンドルは、クローズされません。

戻り値　戻り値はありません。

注：他のレベルでファイルをオープンしている場合（fopenなどでファイルストリームをオープンしている時など）は、この関数を使用してはいけません。

APAGE

レベル０

書　　式

```
# include <iocslib. h>

int APAGE (MODE) ;
int MODE; /*  書き込みページ  */
```

機　　能

グラフィック画面の書き込みページを設定します。
書き込みページは MODE で指定します。

0：書き込みページに０ページを指定
1：書き込みページに１ページを指定
2：書き込みページに２ページを指定
3：書き込みページに３ページを指定
−1：現在の書き込みページを調べる

戻り値

終了コードを返します。

0：正常終了
−1：グラフィックは使用不可
−2：ページ引数が規定外
−3：指定されたページは、現在のモードでは設定不可

MODE が−1のとき正常終了ならば0〜3の値を返します。

B_BADFMT

レベル 0

書　式　# include <iocslib. h>

```
int B_BADFMT (DRIVE, RECNO, MODE) ;
int DRIVE; /* ドライブ番号 */
int RECNO; /* レコード番号 */
int MODE; /* インタリーブコード */
```

機　能　不良トラックを使用不能にします（ハードディスクのみ)。
DRIVE には、pda×256 を指定します。

● pda
　0x80～0x8F　ハードディスク

RECNO には 256 バイト単位のレコード番号を指定します。
MODE にはインタリーブコード（6、1）を指定します（通常 10 メガバイトドライブは 6、20・40 メガバイトドライブは 1)。

戻り値　0xFFFFFF??ならばエラーを表します。
正常終了ならば正の数（0 も）を返します。

B_BPEEK

レベル 0

書　式　# include <iocslib. h>

int B_BPEEK (ADDRESS) ;
unsigned char ＊ADDRESS ; /＊ データ格納領域のアドレス ＊/

機　能　ADDRESSで指定されたアドレスからデータをバイト単位で読み込みます。

戻り値　読み込んだデータを返します。

プログラム例

```
#include        <stdio.h>
#include        <iocslib.h>

static char     mem = 0x10;

main()
{
        char    *addr;          /* アドレス */
        int     rdata;

        addr = &mem;
        rdata = B_BPEEK(addr);

        printf("%x: 0x%02x¥n", addr, rdata);
}
```

B_BPOKE

レベル 0

書　式　# include <iocslib. h>

void B_BPOKE (ADDRESS, DATA) ;
unsigned char ＊ADDRESS ; /＊　データ格納領域のアドレス　＊/
int DATA ; /＊　書き込むデータ　＊/

機　能　ADDRESSで指定されたアドレスへ、データを書き込みます（バイト単位)。

戻り値　戻り値はありません。

プログラム例

```
#include        <stdio.h>
#include        <iocslib.h>

static unsigned char    mem = 0x10;

main()
{
        unsigned char   *addr;

        addr = &mem;
        printf("%x: 0x%02x¥n", addr, mem);
        B_BPOKE(addr, 0xff);
        printf("%x: 0x%02x¥n", addr, mem);
}
```

B_CLR_AL

レベル0

書　式　# include <iocslib. h>

```
void B_CLR_AL ( ) ;
```

機　能　テキスト画面全体をクリアします。
カーソルはホームポジションに移動します。

戻り値　戻り値はありません。

B_CLR_ED

レベル 0

書　　式

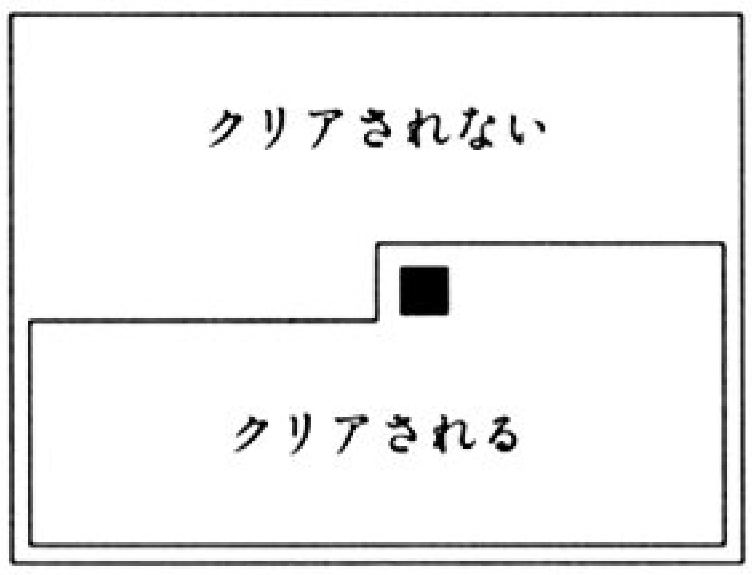
```
# include <iocslib. h>

viod B_CLR_ED ( ) ;
```

機　　能

カーソル位置から最終行右端までテキスト画面をクリアします。

テキスト画面



戻 り 値

戻り値はありません。

B_CLR_ST

レベル 0

書　　式

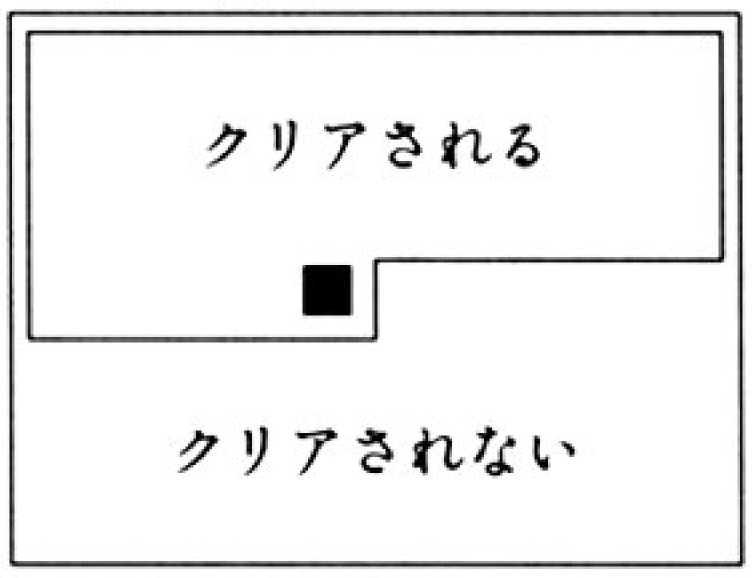
```
# include <iocslib. h>

void B_CLR_ST ( ) ;
```

機　　能

先頭行左端からカーソル位置までテキスト画面をクリアします。

テキスト画面



戻 り 値

戻り値はありません。

B_COLOR　レベル0

書　式

```
# include <iocslib.h>

int B_COLOR (COLOR) ;
int COLOR ; /*  カラー属性  */
```

機　能

COLORで指定した属性を設定します。
COLORに指定できる値とその内容は次の通りです。

0	黒	8	黒
1	水色	9	水色のリバース
2	黄色	10	黄色のリバース
3	白	11	白のリバース
4	黒	12	黒
5	水色の強調	13	水色の強調リバース
6	黄色の強調	14	黄色の強調リバース
7	白の強調	15	白の強調リバース

なお、COLORに−1を指定したときは、属性の通知のみ行います。

戻り値

前の属性を返します。
−1が返ったときは、有効範囲外を指定したことを示します。

B_CONSOL

レベル0

書　式

```
# include <iocslib. h>

int B_CONSOL (XS, YS, XL, YL) ;
int XS; /*  先頭X座標  */
int YS; /*  先頭Y座標  */
int XL; /*  Xの桁数−1  */
int YL; /*  Yの行数−1  */
```

機　能

表示範囲を指定します。
カーソルはホームポジションに移動します。
各座標の指定方法は次の通りです。

XS ： 先頭X座標（16の倍数を指定。0～1008）
YS ： 先頭Y座標（4の倍数を指定。0～1020）
XL ： Xの桁数−1（8ドット単位。0～127）
YL ： Yの桁数−1（16ドット単位。0～63）

ただし、XS・YS両方が−1だった場合、あるいはXL・YL両方が−1だった場合には、先頭座標、桁・行はそれぞれ直前までの値が引きつがれます。

戻り値

上位16ビットには直前のXL、下位16ビットには直前のYLが返されます。

B_CUROFF

レベル０

書　　式　# include <iocslib. h>

```
void B_CUROFF ( ) ;
```

機　　能　カーソルを消します。

戻 り 値　戻り値はありません。

B_CURON

レベル 0

書　　式

```
# include <iocslib. h>

void B_CURON ( ) ;
```

機　　能　カーソルを表示します。

戻 り 値　戻り値はありません。

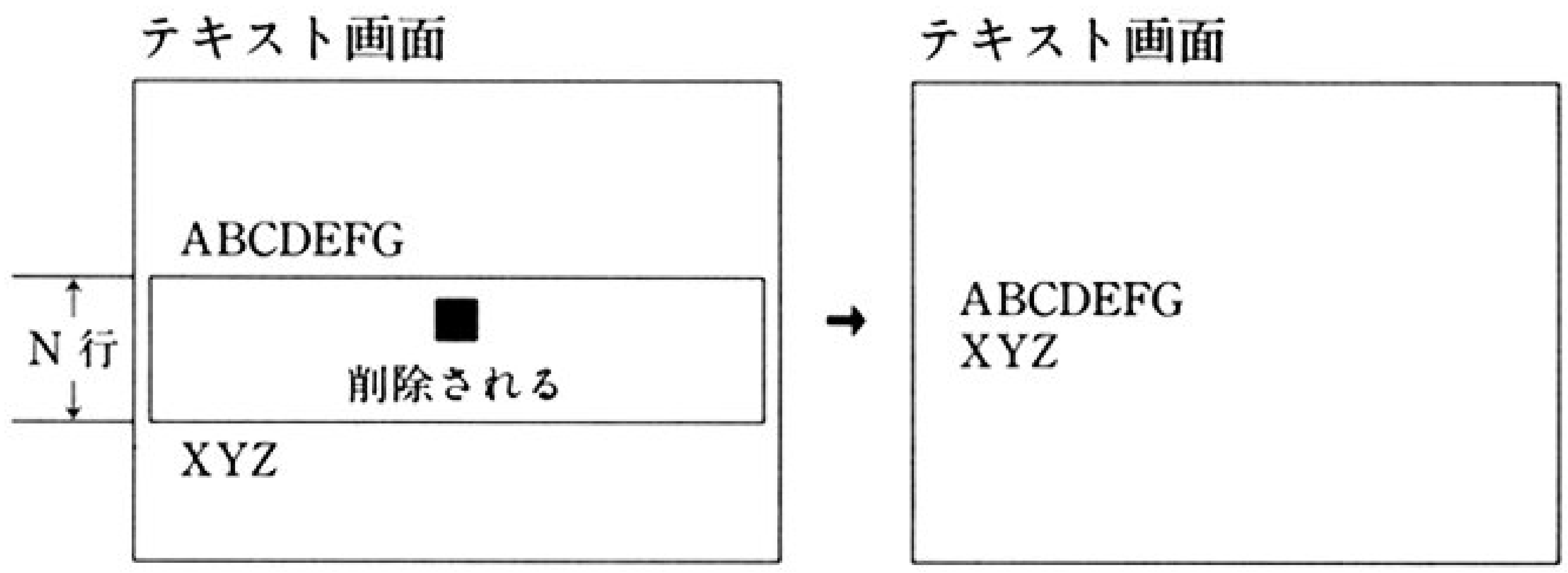
B_DEL

レベル 0

書　式　# include <iocslib. h>

```
void B_DEL (N) ;
int N;  /*  削除する行数  */
```

機　能　カーソル位置の行から N 行削除します。
N に 0 を指定すると、1 とみなされます。
なお、カーソルの X 座標は 0 となります。



戻り値　戻り値はありません。

B_DOWN

レベル 0

書　　式

```
# include <iocslib. h>

void B_DOWN (N) ;
int N ; /*  移動する行数  */
```

機　　能

カーソル位置を、N行下へ移動します。
カーソル位置が最下行を超えてもスクロールアップは行われません。
なお、Nに0を指定すると、1とみなされます。

戻 り 値

戻り値はありません。

B_DOWN_S

レベル０

書　　式　# include <iocslib. h>

```
void B_DOWN_S ( ) ;
```

機　　能　カーソル位置を、１行下へ移動します。
カーソル位置が最下行にあった場合には、スクロールアップが行われます。

戻 り 値　戻り値はありません。

B_DRVCHK

レベル0

書　式

```
# include <iocslib.h>

int B_DRVCHK (DRIVE, MODE) ;
int DRIVE; /* ドライブ番号 */
int MODE; /* モード */
```

機　能

DRIVEで指定したドライブの状態のチェックと設定を行います（2HDのみ）。
DRIVEにはpda×256を指定します。

- pda
  0x90～0x93　2HDフロッピーディスク

MODEには次の内容を設定します。

0：状態のチェックのみ行う（戻り値のビット7～ビット0）
1：イジェクトする（イジェクト禁止状態のときは不可）
2：イジェクト禁止1（MODE ＝ 1のイジェクトも禁止）
3：イジェクト許可1
4：ディスクがセットされていないときLED点滅
5：ディスクがセットされていないときLED消灯
6：イジェクト禁止2（MODE ＝ 1のイジェクトも禁止）
7：イジェクト許可2
8：前回のチェック後にイジェクトしたか否かのチェック

上記の6～8はHuman68Kで使用するので、ユーザーは使用してはいけません。

戻り値

MODE ＝ 0～7のときは、次の内容を返します。

ビット

7	6	5	4	3	2	1	0
LED点滅	EJECT禁止	BUFFER有り	USER使用禁止	PRO	READY	メディア挿入	誤挿入

PRO（プロテクト ＝ 1）、READY（ノットレディ ＝ 1）はMODE ＝ 0で、メディア挿入のときにのみ返します。
MODE ＝ 8のとき、戻り値が1ならばイジェクトしていないことを、−1ならばイジェクトしたことを示します。

B_DRVSNS

レベル0

書　　式

include <iocslib. h>

int B_DRVSNS (DRIVE) ;
int DRIVE ; /* ドライブ番号 */

機　　能

ディスクのステータスを調べます。
DRIVE には pda×256 を指定します。

● pda

0x80～0x8F　ハードディスク
0x90～0x93　2HD フロッピーディスク

戻 り 値

0xFFFFFF??ならばエラーを表します。
正常終了ならばハードディスクは正の数（0も）を、2HD フロッピーディスクは最上位8ビットに FDC のステータス情報（負の数もありえる）を返します。

B_DSKINI

レベル0

〈2HD フロッピーの場合〉

書　式

```
# include <iocslib. h>

int B_DSKINI (DRIVE, DATAADDRESS, OFFTIME) ;
int DRIVE;  /*  ドライブ番号  */
unsigned char *DATAADDRESS;  /*  データアドレス  */
int OFFTIME;  /*  モーターオフまでの時間  */
```

機　能

2HD フロッピーディスクのインターフェイスの初期化を行います。
DRIVE には pda×256 を指定します。

● pda

0x90～0x93　2HD フロッピーディスク

DATAADDRESS には、SPECIFY コマンドのデータアドレスを指定します。
0の場合はデフォルト値（0x03、0xd0、0x10）となります。
OFFTIME にはモーターオフまでの時間（n/100秒）を指定します。
0の場合はデフォルト値（200 … 2秒）となります。

戻り値

0xFFFFFF??ならばエラーを表します。正常終了ならば正の数（0も）を返します。

〈ハードディスクの場合〉

書　式

```
# include <iocslib. h>

int B_DSKINI (DRIVE, DATAADDRESS) ;
int DRIVE;  /*  ドライブ番号  */
unsigned char *DATAADDRESS;  /*  データアドレス  */
```

機　能

ハードディスクのインターフェイスの初期化を行います。
DRIVE には pda×256 を指定します。

● pda

0x80～0x8F　ハードディスク

DATAADDRESS には、ドライブパラメータのデータアドレスを指定します。
0の場合はデフォルト値（0x01、0x01、0x00、0x03、0x01、0x35、0x80、0x00、0x00、0x00）となります。

戻り値

0xFFFFFF??ならばエラーを表します。正常終了ならば正の数（0も）を返します。

B_EJECT

レベル0

書　式

```
# include <iocslib. h>

int B_EJECT (DRIVE) ;
int DRIVE ; /* ドライブ番号 */
```

機　能

イジェクトを行います（イジェクト禁止状態でも実行します）。
ハードディスクの場合は、未使用シリンダへヘッドをシークします。
DRIVE には pda×256 を指定します。

● pda

0x80～0x8F　ハードディスク
0x90～0x93　2HD フロッピーディスク

戻り値

0xFFFFFF??ならばエラーを表します。
正常終了ならば正の数（0 も）を返します。

B_ERA_AL

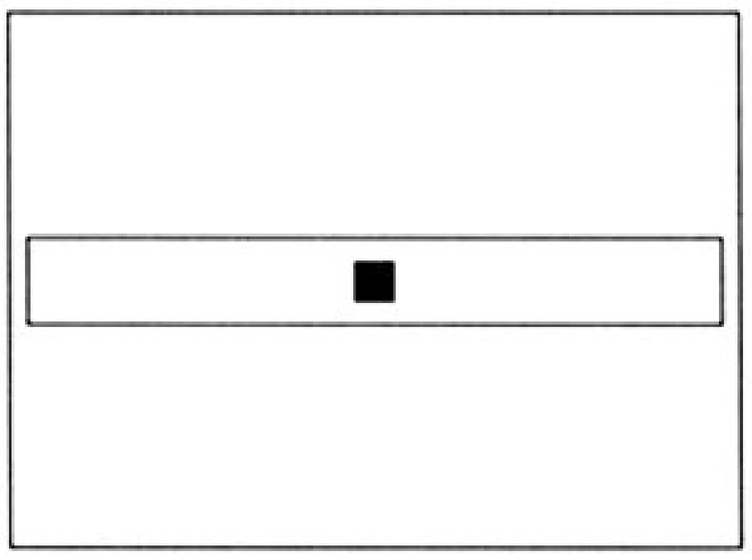
レベル0

書　式　# include <iocslib. h>

void B_ERA_AL ( ) ;

機　能　テキスト画面のカーソルのある行の左端から右端までクリアします。

テキスト画面



□の部分をクリアします。

戻り値　戻り値はありません。

B_ERA_ED

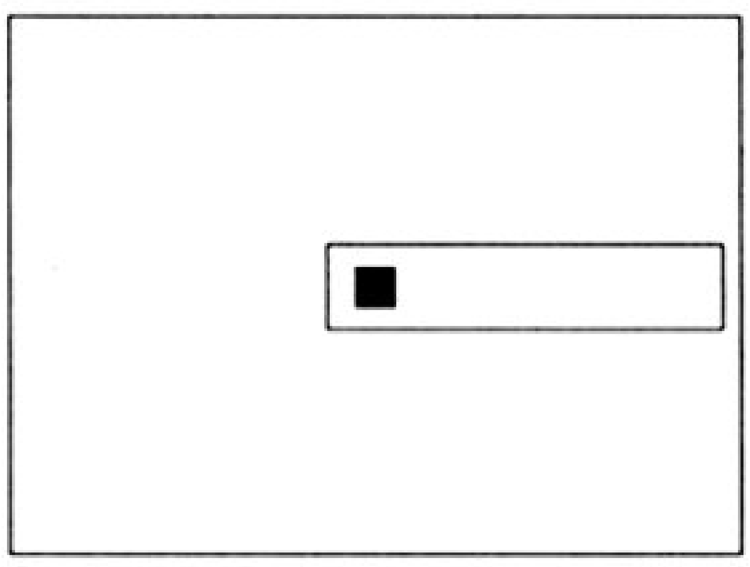
レベル 0

書　式　# include <iocslib. h>

```
void B_ERA_ED ( ) ;
```

機　能　テキスト画面のカーソル位置から、カーソルがある行の右端までをクリアします。

テキスト画面



□の部分をクリアします。

戻り値　戻り値はありません。

B_ERA_ST

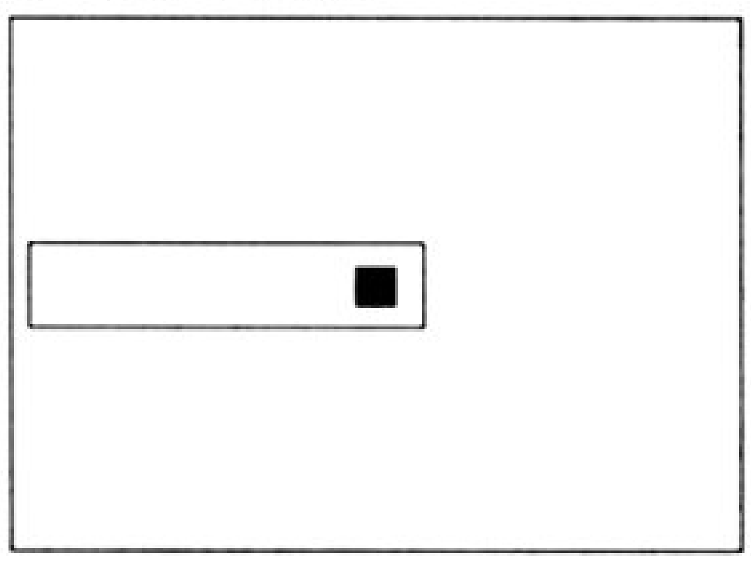
レベル０

書　式　# include 〈iocslib. h〉

void B_ERA_ST ( ) ;

機　能　テキスト画面のカーソルがある行の左端から、カーソル位置までをクリアします。

テキスト画面



□の部分をクリアします。

戻り値　戻り値はありません。

B_FORMAT
レベル 0

書　式

```
# include <iocslib. h>

int B_FORMAT (DRIVE, RECNO, LENGTH, IDADDRESS) ;
int DRIVE; /* ドライブ番号 */
int RECNO; /* レコード番号 */
int LENGTH; /* バイト数 */
unsigned char * IDADDRESS; /* データアドレス */
```

機　能

ディスクの物理フォーマットを行います。

DRIVE には pda×256+mode を指定します。

● pda

0x80～0x8F　ハードディスク

(RECNO は 256 バイト単位のレコード番号となります)

0x90～0x93　2HD フロッピーディスク

(RECNO はセクタ長、トラック、サイド、データの順に上位から 8 ビットずつで指定します)

● mode

mode にはハードディスクの場合、0 を指定します。

2HD フロッピーディスクの場合は、以下のようになります。

0	fm/mfm	retry	seek	0	0	0	0

LENGTH にはバイト数を指定します。

ただしハードディスクの場合は、インタリーブコード (6、1) を指定します (通常 10 メガバイトドライブは 6、20・40 メガバイトドライブは 1)。

IDADDRESS には、ID データの先頭アドレスを指定します。

ただし、ハードディスクの場合は不要です。

IDADDRESS にはスーパーバイザ領域のアドレスも指定できますが、誤ったアドレスを指定しないよう注意してください。

戻り値

0xFFFFFF??ならばエラーを表します。正常終了ならばハードディスクは正の数 (0 も) を、2HD フロッピーディスクは FDC のステータス情報 (負の数もありえる) を返します。

また、2HD フロッピーディスクの場合、物理フォーマットに失敗してもエラーにはならずに、FDC のステータス情報として失敗したことを示す情報が返されます。

BGCTRLGT

レベル 0

書　　式

```
#include <iocslib.h>

int BGCTRLGT (MODE)
int MODE;  /*  モード  */
```

機　　能

BGコントロールレジスタの読み出しを行います。
MODEにはBGコントロールレジスタ番号を指定します（0～1）。

戻り値

テキストページ×2＋表示モード（0～1）、またはエラーコード（－1＝画面モードエラー、画面サイズが768×512のときなど）を返します。

BGCTRLST

レベル 0

書式

```
#include <iocslib.h>

int BGCTRLST (MODE, PAGE, ONOFF) ;
int MODE; /* モード */
int PAGE; /* テキストページ */
int ONOFF; /* 表示の ON/OFF */
```

機能

BG コントロールレジスタの設定を行います。
MODE には BG コントロールレジスタ番号を指定します (0〜1)。
PAGE には、テキストページを指定します (0〜1)。
−1 が指定された場合には、設定を変更せずに現在値を使用します。
ONOFF には表示の ON/OFF を指定します (0 のとき表示 OFF、1 のとき表示 ON)。
−1 が指定された場合には、設定を変更せずに現在値を使用します。

戻り値

終了コードを返します。

0 : 正常終了
−1 : 画面モードエラー (画面サイズが 768×512 のときなど)

BGSCRLGT

レベル 0

書　　式

```
# include <iocslib. h>

int BGSCRLGT (MODE, X, Y) ;
int MODE ; /*  モード  */
int *X ; /*  X 座標を指すポインタ  */
int *Y ; /*  Y 座標を指すポインタ  */
```

機　　能

BG スクロールレジスタの読み出しを行います。
MODE には、BG スクロールレジスタ番号を指定します（0～1）。
X には X 座標、Y には Y 座標を返すべきポインタをそれぞれ指定します。

戻 り 値

終了コードを返します。

　0 ： 正常終了
−1 ： 画面モードエラー（画面サイズが 768×512 のときなど）

正常終了の場合、X 座標、Y 座標がそれぞれのポインタに格納されます。

BGSCRLST

レベル 0

書式 # include <iocslib. h>

```
int BGSCRLST (MODE, X, Y) ;
int MODE; /* 垂直帰線期間の検出モード */
int X; /* X 座標 */
int Y; /* Y 座標 */
```

機能 BG スクロールレジスタの設定を行います。
MODE は次に示すように指定します。

ビット 31 ： 0 垂直帰線期間を検出してから設定を行う
　　　　　： 1 垂直帰線期間を検出しないで設定を行う
ビット 0 ： 0 BG0 を設定する
　　　　： 1 BG1 を設定する

X には X 座標、Y には Y 座標をそれぞれ 0～1023 の範囲内で指定します。
−1 が指定された場合には、設定を変更せずに現在値を使用します。

戻り値 終了コードを返します。

0 ： 正常終了
−1 ： 画面モードエラー（画面サイズが 768×512 のときなど）

BGTEXTCL

レベル 0

書　式

```
#include <iocslib.h>

int BGTEXTCL (PAGE, CODE) ;
int PAGE; /* テキストページ */
int CODE; /* パターンコード */
```

機　能

BGテキストを指定したパターンでクリアします。
PAGEには、テキストページを指定します（0～1）。
CODEには、パターンコードを指定します。
このパターンコードとは、次のV、H、COLOR、PCGCODEの4つのパラメータを各ビットに割り当てたパターンのことです。

ビット15	V：0	縦方向反転しない
	V：1	縦方向反転する
ビット14	H：0	横方向反転しない
	H：1	横方向反転する
ビット8～11	COLOR	パレットブロック(0～15)
ビット0～7	PCGCODE	PCGコード(0～255)

※ビット12、13、16～31は常に0

したがって、CODE＝0x873Cの時は、PCGコードが60、パレットブロックが7、縦方向のみ反転、という設定になります。

戻り値

終了コードを返します。

0：正常終了
−1：画面モードエラー（画面サイズが768×512のときなど）

BGTEXTGT

レベル 0

書　式

```
#include <iocslib.h>

int BGTEXTGT (PAGE, X, Y) ;
int PAGE; /*　テキストページ　*/
int X; /*　テキストX座標　*/
int Y; /*　テキストY座標　*/
```

機　能

BGテキストの読み出しを行います。
PAGEには、テキストページを指定します（0～1）。
Xにはテキスト X 座標、Yにはテキスト Y 座標を、それぞれ0～63の範囲内で指定します。

戻り値

正常終了の時はパターンコードを返します。
パターンコードとは、次の4つのデータを各ビットに割り当てたパターンのことです。

ビット15	V：0	縦方向反転していない
	V：1	縦方向反転している
ビット14	H：0	横方向反転していない
	H：1	横方向反転している
ビット8～11	COLOR	パレットブロック(0～15)
ビット0～7	PCGCODE	PCGコード(0～255)

※ビット12、13、16～31は常に0

異常終了の時はエラーコード（−1）を返します。
これは画面モードが不正（画面サイズが768×512の時など）な場合です。

BGTEXTST

レベル０

書　式

```
#include <iocslib.h>

int BGTEXTST (PAGE, X, Y, CODE) ;
int PAGE; /* テキストページ */
int X; /* テキストX座標 */
int Y; /* テキストY座標 */
int CODE; /* パターンコード */
```

機　能

BGテキストの設定を行います。

PAGEにはテキストページを指定します（0～1）。

Xにはテキスト X座標、Yにはテキスト Y座標をそれぞれ0～63の範囲内で指定します。

CODEにはパターンコードを指定します。

このパターンコードとは、次のV, H, COLOR, PCGCODEの4つのパラメータを各ビットに割り当てたパターンのことです。

ビット15	V：0	縦方向反転しない
	V：1	縦方向反転する
ビット14	H：0	横方向反転しない
	H：1	横方向反転する
ビット8～11	COLOR	パレットブロック(0～15)
ビット0～7	PCGCODE	PCGコード(0～255)

※ビット12、13、16～31は常に0

戻り値

終了コードを返します。

0：正常終了

−1：画面モードエラー（画面サイズが768×512のときなど）

BINDATEBCD　レベル 0

書　式

```
#include <iocslib.h>

int BINDATEBCD (BINDATE) ;
int BINDATE ;  /*  日付  */
```

機　能

2進数の日付を、時計にセットできる形式に変換します。
BINDATEの形式と、設定内容は次の通りです。

0000 yyyy yyyyyyyy mmmmmmmm dddddddd

yyyyyyyyyyyy ： 年 (1980～2079)
mmmmmmmm ： 月 (01～12)
dddddddd ： 日 (01～31)

戻り値

次の内容を返します。

UUUUWWWW YYYYYYYY MMMMMMMM DDDDDDDD

UUUU ： うるう年カウンタ (0～3)
WWWW ： 曜日カウンタ (0～6)　0：日曜日
YYYYYYYY ： 年 (00～99。1980年から相対年数)
MMMMMMMM ： 月 (01～12)
DDDDDDDD ： 日 (01～31)

上記の年，月，日はいずれもBCD2桁です。
エラーが発生したときには−1を返します。

プログラム例

```
#include        <iocslib.h>

#define YEAR            1990
#define MONTH           3
#define DATE            14

main()
{
        int     bin;            /* 日付(2進数) */
        int     bcd;            /* 日付(BCD) */

        bin = (YEAR << 16) | (MONTH << 8) | (DATE << 0);

        bcd = BINDATEBCD(bin);

        BINDATESET(bcd);        /* 日付を時計に設定 */
}
```

BINDATEGET

レベル 0

書　式　#include <iocslib. h>

int BINDATEGET ( ) ;

機　能　時計から日付を読み込みます。

戻り値　次の内容を返します。

0000 WWWW YYYYYYYY MMMMMMMM DDDDDDDD

WWWW	：	曜日カウンタ (0～6)　0：日曜日
YYYYYYYY	：	年 (00～99。1980 年からの相対年数)
MMMMMMMM	：	月 (01～12)
DDDDDDDD	：	日 (01～31)

上記の年，月，日はいずれも BCD2 桁です。

プログラム例

```
#include        <stdio.h>
#include        <iocslib.h>

main()
{
        int     bcd;            /* 日付(BCD) */
        int     bin;            /* 日付(2進数) */

        bcd = BINDATEGET();

        bin = DATEBIN(bcd);

        printf("day   = %d¥n", (bin >> 28) & 0x0f);
        printf("year  = %d¥n", (bin >> 16) & 0x0fff);
        printf("month = %d¥n", (bin >> 8) & 0xff);
        printf("date  = %d¥n", (bin >> 0) & 0xff);
}
```

BINDATESET

レベル 0

書　式　# include <iocslib. h>

void BINDATESET (BCDDATE) ;
int BCDDATE; /* 日付 */

機　能　時計に日付を設定します。
BCDDATE の形式と指定内容は次の通りです。

UUUUWWWW YYYYYYYY MMMMMMMM DDDDDDDD

UUUU : うるう年カウンタ (0～3)
WWWW : 曜日カウンタ (0～6)　0 : 日曜日
YYYYYYYY : 年 (00～99。1980 年からの相対年数)
MMMMMMMM : 月 (01～12)
DDDDDDDD : 日 (01～31)

上記の年，月，日はいずれも BCD2 桁です。

戻り値　戻り値はありません。

プログラム例

```
#include        <iocslib.h>

#define YEAR            1990
#define MONTH           3
#define DATE            14

main()
{
        int     bin;            /* 日付(2進数) */
        int     bcd;            /* 日付(BCD) */

        bin = (YEAR << 16) | (MONTH << 8) | (DATE << 0);

        bcd = BINDATEBCD(bin);

        BINDATESET(bcd);        /* 日付を時計に設定 */
}
```

BINDNO

レベル 1

書　式

include <doslib. h>

int BINDNO (file1, file2) ;
unsigned char ＊file1 ; /＊ 実ファイル名へのポインタ ＊/
unsigned char ＊file2 ; /＊ モジュールファイル名へのポインタ ＊/

機　能

ファイル名からモジュール番号を求めます。
file2 は、file1 で指定されたファイルに含まれる、ファイルのファイル名へのポインタです。
オーバーレイ X ファイルに含まれる個々のファイルを実行するには、各ファイルに割当てられたモジュール番号を指定しなければなりません。
その時は、この BINDNO 関数でモジュール番号を求めることができます。

戻り値

file2 で指定されたファイルのモジュール番号をビット 15～8 に返します。
戻り値が負の場合はエラーコードを示します。

プログラム例

```
#include        <stdio.h>
#include        <stdlib.h>
#include        <doslib.h>


main(int argc, char *argv[])
{
        char    *file;
        char    *module;
        int     mno;

        if (argc < 3) {
                printf("Usage: bindno file module1¥n");
                exit(1);
        }

        file = *++argv;
        --argc;
        while (--argc) {
                module = *++argv;
                if ((mno = BINDNO(file, module)) >= 0) {
                        printf("%s: %s mno = %d¥n",
                                 file, module, mno);
                }
        }
        exit(0);
}
```

B_INS

レベル 0

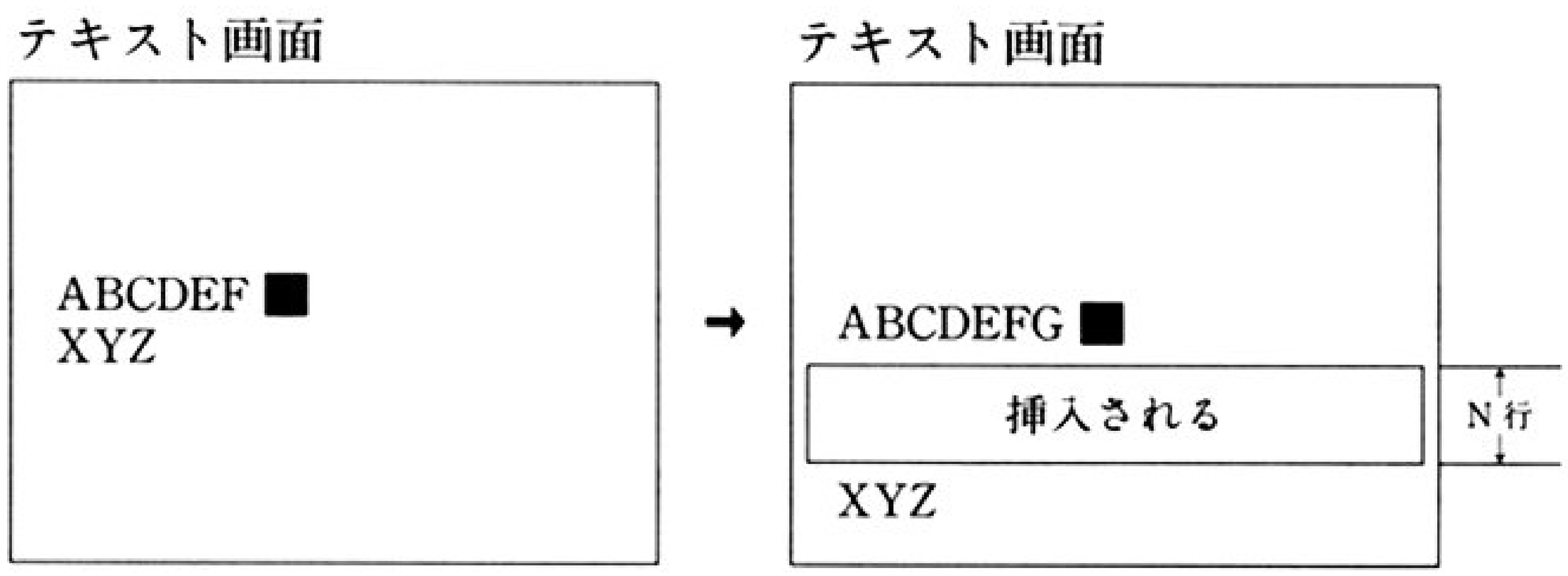
書式

```
# include <iocslib. h>

void B_INS (N) ;
int N;  /*  挿入する行数  */
```

機能

カーソル表示行の直後に N 行挿入します。
N に 0 を指定すると、1 とみなされます。
なお、カーソルの X 座標は 0 となります。



戻り値

戻り値はありません。

B_INTVCS

レベル 0

書　式　# include <iocslib. h>

```
int B_INTVCS (VECTOR, ADDRESS) ;
int VECTOR ;  /*  ベクタ番号  */
int ADDRESS ;  /*  処理ルーチンのエントリアドレス  */
```

機　能　ベクタの設定を行います。

VECTOR にはベクタ番号を指定します。

0～0xFF	:	割り込み
0x100～0x1FF	:	IOCS コール

ADDRESS には、処理ルーチンのエントリアドレスを指定します。

戻り値　前に設定されていた処理ルーチンのエントリアドレスを返します。

BITSNS

レベル0

書　式

```
#include <iocslib. h>

int BITSNS (KEYGRP) ;
int KEYGRP;  /*  キーコードグループ  */
```

機　能

以下に示すキーコードグループをKEYGRPに指定して、キーの押下状態を調べます。

グループ＼ビット	0	1	2	3	4	5	6	7
0	未定義	ESC	1 !	2 ”	3 #	4 $	5 %	6 &
1	7 ’	8 (	9)	0	ー =	^	¥ |	BS
2	TAB	Q	W	E	R	T	Y	U
3	I	O	P	@	[	CR	A	S
4	D	F	G	H	J	K	L	;
5	:	]	Z	X	C	V	B	N
6	M	, <	. >	/ ?	_	SP	HOME	DEL
7	Roll up	Roll down	UNDO	←	↑	→	↓	CLR
8	/	*	ー	7	8	9	+	4
9	5	6	=	1	2	3	ENTER	0
A	,	.	記号	登録	HELP	XF1	XF2	XF3
B	XF4	XF5	かな	ローマ字	コード	CAPS	INS	ひらがな
C	全角	BREAK	COPY	F・1	F・2	F・3	F・4	F・5
D	F・6	F・7	F・8	F・9	F・10	未定義	未定義	未定義
E	SHIFT	CTRL	OPT.1	OPT.2	未定義	未定義	未定義	未定義
F	未定義	未定義	未定義	未定義	未定義	未定義	未定義	未定義

戻り値

各キーコードグループの押下状態を返します。
該当ビットが1ならば、押下されていることを示します。

B_KEYINP

レベル 0

書　　式　# include <iocslib. h>

int B_KEYINP ( ) ;

機　　能　キーコードの読み出しを行います。
また、電卓が処理されます。

戻 り 値　スキャンコード×256＋内部コードを返します。
スキャンコードは、BITSNS 関数で指定するキーコードグループを 8 倍したものに、そのグループのキーのビット位置の番号を加えたものです。
例えば、'A'が押されていたならば、グループは 3、ビット位置は 6 なので、3×8+6=0x1E になります。

プログラム例

```
#include        <stdio.h>
#include        <iocslib.h>

main()
{
        int     key_code;                /* キーコード */

        key_code = B_KEYINP();
        printf("key_code      = %d¥n", key_code);
        printf("scan code     = 0x%02x¥n", key_code / 256);
        printf("internal code = 0x%02x¥n", key_code % 256);
        printf("character     = %c¥n", key_code % 256);
}
```

B_KEYSNS

レベル 0

書　　式

```
# include <iocslib. h>

int B_KEYSNS ( ) ;
```

機　　能

キーの先行入力があるかないかを調べます。
また、電卓が処理されます。

戻 り 値

0x010000＋スキャンコード×256＋内部コードを返します。
0 の場合には、キーが押下されていないことを示します。
スキャンコードは、B_KEYINP 関数を参照してください。

B_LEFT

レベル 0

書　　式

```
# include <iocslib. h>

void B_LEFT (N)
int N ; /*  移動する桁数  */
```

機　　能

カーソル位置をN桁左へ移動します。
カーソル位置が左端を超えても、右スクロールは行われません。
なお、Nに0を指定すると、1とみなされます。

戻 り 値

戻り値はありません。

B_LOCATE

レベル0

書式

```
# include <iocslib. h>

int B_LOCATE (X, Y) ;
int X;  /*  X座標  */
int Y;  /*  Y座標  */
```

機能

XとYで指定した位置にカーソルを設定します。

Xに−1を指定した場合には、前の座標の通知のみ行います。

戻り値

上位16ビットに前のX座標、下位の16ビットに前のY座標を返します。

B_LPEEK

レベル０

書　式　# include <iocslib. h>

int B_LPEEK (ADDRESS) ;
unsigned long ＊ADDRESS ; /＊ データの格納領域のアドレス ＊/

機　能　ADDRESSで指定されたアドレスから、データをロングワード単位（4バイト）で読み込みます。
このときアドレスは偶数でなければなりません。

戻り値　読み込んだデータを返します。

プログラム例

```
#include        <stdio.h>
#include        <iocslib.h>

static unsigned long    mem = 0x100;

main()
{
        unsigned long   *addr;          /* アドレス */
        int     rdata;                  /* 読み込んだデータ */

        addr = &mem;

        rdata = B_LPEEK(addr);
        printf("%x: 0x%04x¥n", addr, rdata);
}
```

B_LPOKE

レベル 0

書　　式

include <iocslib. h>

void B_LPOKE (ADDRESS, DATA) ;
unsigned long ＊ADDRESS; /＊　データ格納領域のアドレス　＊/
int DATA; /＊　書き込むデータ　＊/

機　　能

ADDRESSで指定されたアドレスへ、データをロングワード単位（4バイト）で書き込みます。
このときアドレスは偶数でなければなりません。

戻 り 値

戻り値はありません。

プログラム例

```
#include        <stdio.h>
#include        <iocslib.h>

static unsigned long    mem = 0x100;

main()
{
        unsigned long   *addr;  /* データ書込み領域アドレス */

        addr = &mem;
        printf("%x: 0x%04x¥n", addr, *addr);

        B_LPOKE(addr, 0x10000);
        printf("%x: 0x%04x¥n", addr, *addr);
}
```

B_MEMSET

レベル０

書　式

\# include <iocslib. h>

void B_MEMSET (ADDRESS, DATABUF, DATABYTE) ;
unsigned char ＊ADDRESS ; /＊　データ書き込み領域のアドレス　＊/
unsigned char ＊DATABUF ; /＊　データ格納領域のアドレス　＊/
int DATABYTE ; /＊　データ書き込みバイト数　＊/

機　能

DATABUF が指す領域のデータを、ADDRESS で指定されたアドレスへ書き込みます。
DATABYTE には、書き込むデータのバイト数−1 を指定します。
このとき 2 つの領域を重ねてはいけません。

戻り値

戻り値はありません。

プログラム例

```
#include        <stdio.h>
#include        <iocslib.h>

#define DATA_SIZE       100

main()
{
        unsigned char   src[DATA_SIZE]; /* 元データ */
        unsigned char   des[DATA_SIZE]; /* 転送先 */
        int     i;

        for (i = 0; i < DATA_SIZE; i++)
                src[i] = i;

        B_MEMSET(des, src, DATA_SIZE - 1);

        for (i = 0; i < DATA_SIZE; i++) {
                if (i % 10 == 9)
                        printf("%02x¥n", des[i]);
                else
                        printf("%02x ", des[i]);
        }
}
```

B_MEMSTR

レベル 0

書　式　# include <iocslib. h>

void B_MEMSTR (ADDRESS, DATABUF, DATABYTE) ;
unsigned char ＊ADDRESS; /＊ データ読み込み領域のアドレス ＊/
unsigned char ＊DATABUF; /＊ データ格納領域のアドレス ＊/
int DATABYTE; /＊ データ読み込みバイト数 ＊/

機　能　ADDRESSで指定されたアドレスからデータを読み込み、DATABUFが指す領域へ格納します。
DATABYTEには、読み込むデータのバイト数－1を指定します。
このとき2つの領域を重ねてはいけません。

戻り値　戻り値はありません。

プログラム例

```
#include        <stdio.h>
#include        <iocslib.h>

#define DATA_SIZE       100

main()
{
        unsigned char   src[DATA_SIZE]; /* 元データ */
        unsigned char   des[DATA_SIZE]; /* 転送先 */
        int     i;

        for (i = 0; i < DATA_SIZE; i++)
                src[i] = i;

        B_MEMSTR(src, des, DATA_SIZE - 1);

        for (i = 0; i < DATA_SIZE; i++) {
                if (i % 10 == 9)
                        printf("%02x¥n", des[i]);
                else
                        printf("%02x ", des[i]);
        }
}
```

BOOTINF

レベル 0

書式

```
# include <iocslib. h>
int BOOTINF( );
```

機能

パワーON情報とシステムのブート情報を返します。

戻り値

以下に示すブート情報を返します。

ビット31～24：パワーON情報
- 0x00：パワースイッチにより起動
- 0x01：外部スイッチにより起動
- 0x02：タイマにより起動

ビット23～0：システムのブート情報
- 0x000080～0x00008F：ハードディスクより起動（下位4ビットがブートしたドライブの番号を表す）
- 0x000090～0x000093：2HDディスクより起動（下位2ビットがブートしたドライブの番号を表す）
- 0xED0000～0xED3FFF：SRAMからブートしたときのブートアドレス
- 上記以外の値：ROMからブートしたときのブートアドレス

プログラム例

```
#include        <stdio.h>
#include        <iocslib.h>


main()
{
        long    inf;
        int     pow;
        long    bot;

        inf = BOOTINF();

        pow = (inf >> 24) & 0xff;
        switch (pow) {
        case    0x00:
                printf("パワースイッチにより起動¥n");
                break;
        case    0x01:
                printf("外部スイッチにより起動¥n");
                break;
        case    0x02:
                printf("タイマにより起動¥n");
                break;
        }

        bot = inf & 0x00ffffff;
        if (0x000080 <= bot && bot <= 0x00008f)
                printf("boot from ハードディスク¥n");
        else if (0x000090 <= bot && bot <= 0x000093)
                printf("boot from floppyディスク¥n");
        else if (0xed0000 <= bot && bot <= 0xed3fff)
                printf("boot from SRAM¥n");
        else
                printf("boot from ROM¥n");

}
```

BOX

レベル 0

書　　式

```
#include <iocslib.h>

int BOX (ptr) ;
struct BOXPTR {
  short x1 ; /* 始点の X 座標 */
  short y1 ; /* 始点の Y 座標 */
  short x2 ; /* 終点の X 座標 */
  short y2 ; /* 終点の Y 座標 */
  unsigned short color ; /* パレットコード */
  unsigned short linestyle ; /* ラインスタイル */
} *ptr ;
```

機　　能

グラフィック画面にボックスを描画します。

戻 り 値

終了コードを返します。

　0 ： 正常終了
−1 ： グラフィックは使用不可

B_PRINT

レベル0

書　　式

```
# include <iocslib. h>

int B_PRINT (MSGPTR) ;
unsigned char *MSGPTR; /*　文字列データを指すポインタ　*/
```

機　　能

ヌル文字で終了する文字列データを表示します。

戻 り 値

上位16ビットに表示後のカーソルX座標、下位16ビットにカーソルY座標を返します。

B_PUTC

レベル 0

書　式

```
# include <iocslib. h>

int B_PUTC (CODE) ;
int CODE;  /*  表示するデータ  */
```

機　能　CODEで指定した1バイトデータを表示します。

戻り値　上位16ビットに表示後のカーソルX座標、下位16ビットにカーソルY座標を返します。

B_PUTMES

レベル 0

書　式

```
# include <iocslib. h>

int B_PUTMES (COLOR, X, Y, MAX, ADDRESS) ;
int COLOR ; /*  属性  */
int X ; /*  X座標  */
int Y ; /*  Y座標  */
int MAX ; /*  最大表示文字数－1  */
unsigned char *ADDRESS ; /*  表示文字列を指すポインタ  */
```

機　能

指定位置（絶対座標）に、指定された文字列を表示します。
指定文字数を超えて表示されることはありません。
表示しようとした文字列が、全角文字の前半で切られてしまうような場合は、その全角文字は表示されず、空白が表示されます。
指定した文字数分表示する前にヌル文字が現れたら、以降の文字は空白で埋められます。
この関数は、ファンクション行表示用であり、B_CONSOLには影響されません。また、表示後のカーソル位置にも変化はありません。
COLORには属性を指定します。
XにはX座標（絶対座標）、YにはY座標（絶対座標）をそれぞれ指定します。
また、MAXには文字数－1（半角単位）を、ADDRESSには、ヌル文字で終了する文字列の先頭アドレスを指定します。

戻り値

次のX座標の値を返します。

B_READ

レベル 0

書式

```
# include <iocslib. h>
int B_READ (DRIVE, RECNO, LENGTH, ADDRESS) ;
int DRIVE; /* ドライブ番号 */
int RECNO; /* レコード番号 */
int LENGTH; /* バイト数 */
unsigned char * ADDRESS; /* データ読み込み領域の先頭アドレス */
```

機能

ディスクからデータを読み込みます。

DRIVE には pda×256＋mode を指定します。

- pda

  0x80～0x8F ：ハードディスク

  (RECNO は 256 バイト単位のレコード番号となります)

  0x90～0x93 ：2HD フロッピーディスク

  (RECNO はセクタ長、トラック、サイド、セクタの順に上位から 8 ビットずつ設定します)

- mode

  mode にはハードディスクの場合、0 を指定します。

  2HD フロッピーディスクの場合は以下のようになります。

0	fm/mfm	retry	seek	0	0	0	0

LENGTH には、読み込みを行うデータのバイト数を指定します。

ハードディスクの場合は 256 の倍数を、2HD フロッピーディスクの場合は、1024 の倍数を指定してください。

ADDRESS には、データ読み込み領域の先頭アドレスを指定します。

ADDRESS にはスーパーバイザ領域のアドレスも指定できますが、誤ったアドレスを指定しないよう注意してください。

戻り値

0xFFFFFF??ならばエラーを表します。正常終了ならばハードディスクは正の数（0 も）を、2HD フロッピーディスクは FDC のステータス情報（負の数もありえる）を返します。

また、2HD フロッピーディスクの場合、読み込みに失敗してもエラーにはならずに、FDC のステータス情報として失敗したことを示す情報が返されます。

B_READDI

レベル0

書　式

```
# include <iocslib. h>
int B_READDI (DRIVE, RECNO, LENGTH, ADDRESS) ;
int DRIVE;  /*  ドライブ番号  */
int RECNO;  /*  レコード番号  */
int LENGTH;  /*  バイト数  */
unsigned char *ADDRESS;  /*  データ読み込み領域の先頭アドレス  */
```

機　能

診断のための読み出しを行います（2HD のみ)。

DRIVE には pda×256＋mode を指定します。

- pda

  0x90～0x93：2HD フロッピーディスク

  (RECNO はセクタ長、トラック、サイド、セクタの順に上位から8ビットずつ指定します)

- mode

0	fm/mfm	retry	seek	0	0	0	0

LENGTH には読み出しを行うデータのバイト数を指定します。

1024 の倍数を指定してください。

ADDRESS には、データ読み込み領域の先頭アドレスを指定します。

ADDRESS には、スーパーバイザ領域のアドレスも指定できますが、誤ったアドレスを指定しないよう注意してください。

戻り値

0xFFFFFF??ならばエラーを表します。

正常終了ならば FDC のステータス情報（負の数もありえる）を返します。

また、読み込みに失敗してもエラーにはならずに、FDC のステータス情報として失敗したことを示す情報が返されます。

なお、CRC エラーなどが発生しても途中で中断して終了したりせずに、必ず指定バイト数の読み込みを行います。

B_READDL

レベル 0

書　式

```
# include <iocslib. h>

int B_READDL (DRIVE, RECNO, LENGTH, ADDRESS) ;
int DRIVE ;  /*  ドライブ番号  */
int RECNO ;  /*  レコード番号  */
int LENGTH ;  /*  バイト数  */
unsigned char * ADDRESS ;  /*  データ読み込み領域の先頭アドレス  */
```

機　能

削除データを読み込みます（2HDのみ）。
DRIVEにはpda×256+modeを指定します。

● pda

0x90～0x93：2HDフロッピーディスク
(RECNOはセクタ長、トラック、サイド、セクタの順に上位から8ビットずつ設定します)

● mode

0	fm/mfm	retry	seek	0	0	0	0

LENGTHにはデータのバイト数を指定します。
1024の倍数を指定してください。
ADDRESSには、データ読み込み領域の先頭アドレスを指定します。
ADDRESSには、スーパーバイザ領域のアドレスも指定できますが、誤ったアドレスを指定しないよう注意してください。

戻り値

0xFFFFFF??ならばエラーを表します。正常終了ならばFDCのステータス情報（負の数もありえる）を返します。
また、読み込みに失敗してもエラーにはならずに、FDCのステータス情報として失敗したことを示す情報が返されます。

BREAKCK

レベル 1

書　式

\# include <doslib. h>

int BREAKCK (flag) ;
int flag;　/*　ブレークチェックの設定フラグ　*/

機　能

ブレークチェックの設定を行います。
flag に指定できる内容は次の通りです。

0 : 特定のファンクションコールでのみチェックする
1 : すべてのファンクションコールでチェックする
−1 : 設定状況の通知要求

戻り値

設定状況を表す次の値を返します。

0 : 特定のファンクションコールでのみチェックする
1 : すべてのファンクションコールでチェックする

プログラム例

```
#include        <stdio.h>
#include        <doslib.h>

#define CHECK1  0       /* 指定のものをcheckするように設定 */
#define ALL_CK  1       /* すべてcheckするように設定 */
#define STATUS  (-1)    /* 設定状況のセンス */

main()
{
        if (BREAKCK(STATUS) == ALL_CK)
                puts("すべてのファンクションコールをチェック");
        else
                puts("指定のファンクションコールのみチェック");
}
```

B_RECALI

レベル 0

書式

```
# include <iocslib.h>

int B_RECALI (DRIVE) ;
int DRIVE; /* ドライブ番号 */
```

機能

トラック0へシークを行います。
DRIVEにはpda×256を指定します。
pda×256+255のときは、強制レディ状態での調査を行います（2HDのときのみ）。

● pda
　0x80～0x8F：ハードディスク
　0x90～0x93：2HDフロッピーディスク

戻り値

0xFFFFFF??ならばエラーを表します。正常終了ならばハードディスクは正の数（0も）を、2HDフロッピーディスクはFDCのステータス情報（負の数もありえる）を返します。
また、2HDフロッピーディスクの場合、シーク動作に失敗してもエラーにはならずに、FDCのステータス情報として失敗したことを示す情報が返されます。

B_RIGHT

レベル0

書　式

```
# include <iocslib. h>

void B_RIGHT (N) ;
int N; /* 移動する桁数 */
```

機　能

カーソル位置をN桁右へ移動します。
カーソル位置が右端を超えても、左スクロールは行われません。
なお、Nに0を指定すると、1とみなされます。

戻り値

戻り値はありません。

B_SEEK

レベル０

書　式

```
# include <iocslib. h>

int B_SEEK (DRIVE, RECNO) ;
int DRIVE ;  /*  ドライブ番号  */
int RECNO ;  /*  レコード番号  */
```

機　能

指定トラック（レコード）までシークします。
DRIVE には pda×256＋mode を指定します。

● pda

0x80～0x8F：ハードディスク
(RECNO は 256 バイト単位のレコード番号となります)
0x90～0x93：2HD フロッピーディスク
(RECNO はセクタ長、トラック、サイド、セクタの順に上位から 8 ビットずつ設定します)

● mode

mode にはハードディスクの場合、0 を指定します。
2HD フロッピーディスクの場合は以下のようになります。

0	fm/mfm	retry	seek	0	0	0	0

戻り値

0xFFFFFF??ならばエラーを表します。正常終了ならばハードディスクは正の数（0 も）を、2HD フロッピーディスクは FDC のステータス情報（負の数もありえる）を返します。
また、2HD フロッピーディスクの場合、シーク動作に失敗してもエラーにはならずに、FDC のステータス情報として失敗したことを示す情報が返されます。

B_SFTSNS

レベル0

書　　式　# include <iocslib. h>

int B_SFTSNS ( ) ;

機　　能　シフトキーの押下状態を調べます。

戻 り 値　シフトキーの押下状態を返します。
押下状態を示す各ビットの内容は次の通りで、ビット ＝ 1ならば各シフトキーが押下されたことを示します。

●ビット 15〜8

15	14	13	12	11	10	9	8
0	全角	ひらがな	INS	CAPS	コード入力	ローマ字	かな

●ビット 7〜0

7	6	5	4	3	2	1	0
CAPS	コード入力	ローマ字	かな	OPT2	OPT1	CTRL	SHIFT

上記のうち、LEDの付いているキーは、LEDの点灯・消灯の状態を、LEDの付いていないキーは実際に押下されているかどうかの状態を示します。

B_SUPER

レベル 0

書　式

```
# include <iocslib.h>

int B_SUPER (SSP) ;
int SSP;
```

機　能

スーパーバイザモード、ユーザーモード間の切り替えを行います。
SSP に 0 を指定するとユーザーモードからスーパーバイザモードに替わります。
SSP に 0 以外を指定するとスーパーバイザモードからユーザーモードに替わります。

戻り値

前の SSP を返します。
すでにスーパーバイザモードとなっている状態で、再度スーパーバイザモードに切り替えを行うように指定するとエラーになり、負の数を返します。

B_UP

レベル０

書　式

```
# include <iocslib. h>

void B_UP (N) ;
int N ; /*  移動する行数  */
```

機　能

カーソル位置をＮ行上へ移動します。
カーソル位置が先頭行を超えてもスクロールダウンは行われません。
なお、Ｎに０を指定すると、１とみなされます。

戻り値

戻り値はありません。

B_UP_S

レベル０

書　式

```
# include <iocslib. h>

void B_UP_S ( ) ;
```

機　能

カーソル位置を１行上へ移動します。
カーソル位置が先頭行にあった場合には、スクロールダウンが行われます。

戻り値

戻り値はありません。

BUS_ERR

レベル 1

書式

```
# include <doslib.h>

int BUS_ERR(s_adr, d_adr, mode) ;
unsigned char *s_adr; /* 読み込みポインタ */
unsigned char *d_adr; /* 書き込みポインタ */
int mode; /* アクセスする時の単位サイズ */
```

機能

スーパーバイザ領域や、何もマッピングされていない領域など、アクセスした時にバスエラーが発生する可能性のある領域を、読み書きできるかどうかをチェックして、その結果を返します。
s_adr には、読み込みたい領域へのポインタを指定します。
d_adr には、書き込みたい領域へのポインタを指定します。
mode は、アクセスする時の単位サイズを次のように設定します。

1:バイトでアクセス
2:ワードでアクセス
4:ロングワードでアクセス

戻り値

戻り値が0なら、読み書き両方とも可能であることを示し、読み込みポインタで示されるアドレスからデータを読み込んで、書き込みポインタで示されるポインタへデータを書き込んだことを示します。
1なら、d_adr で指定したアドレスに書き込みを行ったときにバスエラーが発生することを示します。
2なら、s_adr で指定したアドレスで読み込みを行ったときにバスエラーが発生することを示します。
−1なら、おかしな引数を指定したことを示します(mode が不正、あるいは奇数アドレスからワード、ロングワードでアクセスするよう指定したとき)。
s_adr を先にチェックしますので、s_adr, d_adr 両方でバスエラーが発生する場合には2を返します。

プログラム例

```
#include        <stdio.h>
#include        <doslib.h>

#define M_BYTE          1
#define M_WORD          2
#define M_LONG          4
#define R_OK            0
#define R_WRITE         1
#define R_READ          2
#define R_ERR           (-1)


main()
{
        char    *s_adr;
        char    *d_adr;
        int     mode;
        int     r;

        mode = M_BYTE;
        s_adr = (char *)0xc00000;
        d_adr = (char *)0xe00000;
        r = BUS_ERR(s_adr, d_adr, mode);
        switch (r) {
        case R_OK:
                printf("success¥n");
                break;
        case R_WRITE:
                printf("can't write¥n");
                break;
        case R_READ:
                printf("can't read¥n");
                break;
        case R_ERR:
                printf("error¥n");
                break;
        }
}
```

B_VERIFY

レベル 0

書　式

```
# include <iocslib. h>

int B_VERIFY (DRIVE, RECNO, LENGTH, ADDRESS) ;
int DRIVE; /*  ドライブ番号  */
int RECNO; /*  レコード番号  */
int LENGTH; /*  バイト数  */
unsigned char * ADDRESS; /*  比較するデータの先頭アドレス  */
```

機　能

データの比較チェックを行います。
DRIVE には pda×256＋mode を指定します。

● pda

0x80～0x8F：ハードディスク
(RECNO は 256 バイト単位のレコード番号となります)
0x90～0x93：2HD フロッピーディスク
(RECNO はセクタ長、トラック、サイド、セクタの順に上位から 8 ビットずつ設定します)

● mode

mode にはハードディスクの場合、0 を指定します。
2HD フロッピーディスクの場合は以下のようになります。

0	fm/mfm	retry	seek	0	0	0	0

LENGTH には比較チェックを行うデータのバイト数を指定します。ハードディスクの場合は 256 の倍数を、2HD フロッピーディスクの場合は 1024 の倍数を指定してください。
ADDRESS には、比較するデータの先頭アドレスを指定します。
ADDRESS には、スーパーバイザ領域のアドレスも指定できますが、誤ったアドレスを指定しないよう注意してください。

戻り値

0xFFFFFF??ならばエラーを表します。正常終了ならばハードディスクは正の数（0 も）を、2HD フロッピーディスクは FDC のステータス情報（負の数もありえる）を返します。
また、2HD フロッピーディスクの場合、比較チェックに失敗してもエラーにはならずに、FDC のステータス情報として失敗したことを示す情報が返されます。

B_WPEEK

レベル 0

書　式　# include <iocslib. h>

int B_WPEEK (ADDRESS) ;
unsigned short ＊ADDRESS ; /＊ データ格納領域のアドレス ＊/

機　能　ADDRESSで指定されたアドレスから、データをワード単位（2バイト）で読み込みます。
このときアドレスは偶数でなければなりません。

戻り値　読み込んだデータを返します。

プログラム例

```
#include        <stdio.h>
#include        <iocslib.h>

static unsigned short   mem = 0x2000;

main()
{
        unsigned short  *addr;          /* アドレス */
        int     rdata;                  /* 読み込んだデータ */

        addr = &mem;

        rdata = B_WPEEK(addr);
        printf("%x: 0x%02x¥n", addr, rdata);
}
```

B_WPOKE

レベル0

書　式

#include <iocslib. h>

void B_WPOKE (ADDRESS, DATA) ;
unsigned short ＊ADDRESS ; /＊ データ格納領域のアドレス ＊/
int DATA ; /＊ 書き込むデータ ＊/

機　能

ADDRESSで指定されたアドレスへ、データをワード単位（2バイト）で書き込みます。
このときアドレスは偶数でなければなりません。

戻り値

戻り値はありません。

プログラム例

```
#include        <stdio.h>
#include        <iocslib.h>

static unsigned short   mem = 0x100;

main()
{
        unsigned short  *addr;          /* アドレス */
        int     rdata;                  /* 読み込んだデータ */

        addr = &mem;

        rdata = B_WPEEK(addr);
        printf("%x: 0x%02x¥n", addr, rdata);

        B_WPOKE(addr, 0x2000);

        rdata = B_WPEEK(addr);
        printf("%x: 0x%02x¥n", addr, rdata);
}
```

B_WRITE

レベル 0

書　式

```
#include <iocslib.h>

int B_WRITE (DRIVE, RECNO, LENGTH, ADDRESS) ;
int DRIVE;  /*  ドライブ番号  */
int RECNO;  /*  レコード番号  */
int LENGTH;  /*  バイト数  */
unsigned char * ADDRESS;  /*  書き出すデータの先頭アドレス  */
```

機　能

ディスクにデータを書き出します。
DRIVE には pda×256+mode を指定します。

● pda

0x80～0x8F：ハードディスク
(RECNO は 256 バイト単位のレコード番号となります)
0x90～0x93：2HD フロッピーディスク
(RECNO はセクタ長、トラック、サイド、セクタの順に上位から 8 ビットずつ設定します)

● mode

mode にはハードディスクの場合、0 を指定します。
2HD フロッピーディスクの場合は以下のようになります。

0	fm/mfm	retry	seek	0	0	0	0

LENGTH には書き出しを行うデータのバイト数を指定します。ハードディスクの場合は 256 の倍数を、2HD フロッピーディスクの場合は 1024 の倍数を指定してください。
ADDRESS には、書き出すデータの先頭アドレスを指定します。
ADDRESS には、スーパーバイザ領域のアドレスも指定できますが、誤ったアドレスを指定しないよう注意してください。

戻り値

0xFFFFFF??ならばエラーを表します。正常終了ならばハードディスクは正の数（0 も）を、2HD フロッピーディスクは FDC のステータス情報（負の数もありえる）を返します。
また、2HD フロッピーディスクの場合、書き出しに失敗してもエラーにはならずに、FDC のステータス情報として失敗したことを示す情報が返されます。

B_WRITED

レベル0

書式

```
#include <iocslib.h>

int B_WRITED (DRIVE, RECNO, LENGTH, ADDRESS) ;
int DRIVE; /* ドライブ番号 */
int RECNO; /* レコード番号 */
int LENGTH; /* バイト数 */
unsigned char *ADDRESS; /* 書き込みデータの先頭アドレス */
```

機能

削除データを書き込みます（2HDのみ）。
DRIVEにはpda×256+modeを指定します。

● pda

0x90～0x93：2HDフロッピーディスク
（RECNOはセクタ長、トラック、サイド、セクタの順に上位から8ビットずつ設定します）

● mode

0	fm/mfm	retry	seek	0	0	0	0

LENGTHにはデータのバイト数を指定します。1024の倍数を指定してください。
ADDRESSには、書き込みデータの先頭アドレスを指定します。
ADDRESSには、スーパーバイザ領域のアドレスも指定できますが、誤ったアドレスを指定しないよう注意してください。

戻り値

0xFFFFFF??ならばエラーを表します。正常終了ならばFDCのステータス情報（負の数もありえる）を返します。
また、書き込みに失敗してもエラーにはならずに、FDCのステータス情報として失敗したことを示す情報が返されます。

C_CLS_AL

レベル 1

書　式　#include <doslib.h>

```
int C_CLS_AL ( ) ;
```

機　能　テキスト画面全体をクリアします。
カーソルはホームポジションへ移動します。
このファンクションコールは、CON デバイスによりサポートされます。

戻り値　処理が正常に終了したときは、0を返します。

参照関数　C_CLS_ED, C_CLS_ST

C_CLS_ED

レベル 1

書　式

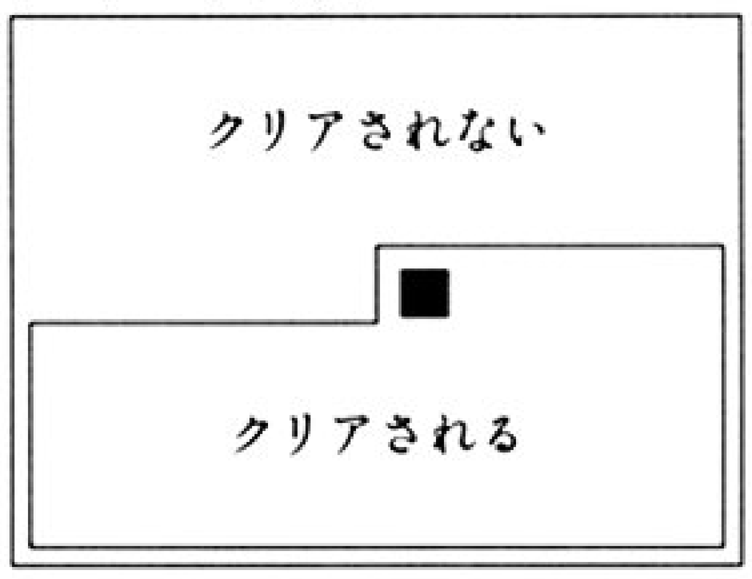
```
#include <doslib.h>

int C_CLS_ED ( ) ;
```

機　能

カーソル位置から最終行右端まで、テキスト画面をクリアします。
このファンクションコールは、CON デバイスによりサポートされます。

テキスト画面



戻 り 値

処理が正常に終了したときは、0を返します。

参照関数

C_CLS_ST, C_CLS_AL

C_CLS_ST

レベル 1

書　式

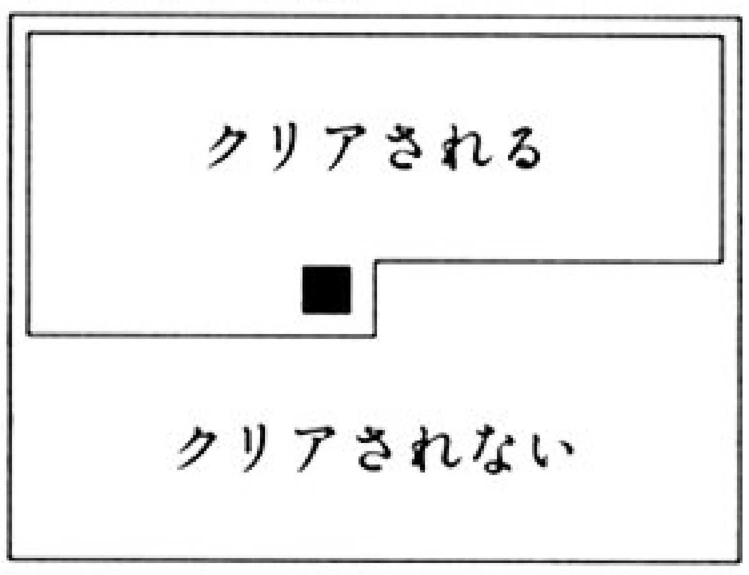
```
#include <doslib.h>

int C_CLS_ST ( ) ;
```

機　能

先頭行左端からカーソル位置まで、テキスト画面をクリアします。
このファンクションコールは、CON デバイスによりサポートされます。

テキスト画面



戻 り 値

処理が正常に終了したときは、0 を返します。

参照関数

C_CLS_ED, C_CLS_AL

C_COLOR

レベル 1

書　式

```
#include <doslib.h>

int C_COLOR (atr) ;
int atr ; /*  カラー属性  */
```

機　能

atrで指定した属性を設定します。
カラー属性の内容は次の通りです。

0	黒	8	黒
1	水色	9	水色のリバース
2	黄色	10	黄色のリバース
3	白	11	白のリバース
4	黒	12	黒
5	水色の強調	13	水色の強調・リバース
6	黄色の強調	14	黄色の強調・リバース
7	白の強調	15	白の強調・リバース

このファンクションコールは、CONデバイスによりサポートされます。

戻り値

処理が正常に終了したときは、0を返します。

C_CUROFF

レベル 1

書　式　# include 〈doslib. h〉

```
int C_CUROFF ( ) ;
```

機　能　カーソルを非表示モードにします。
このファンクションコールは、CON デバイスによりサポートされます。

戻り値　処理が正常に終了したときは、0 を返します。

参照関数　C_CURON

C_CURON

レベル 1

書　　式

```
#include <doslib.h>

int C_CURON ( ) ;
```

機　　能

カーソルを表示モードにします。
このファンクションコールは、CON デバイスによりサポートされます。

戻 り 値

処理が正常に終了したときは、0を返します。

参照関数

C_CUROFF

C_DEL

レベル 1

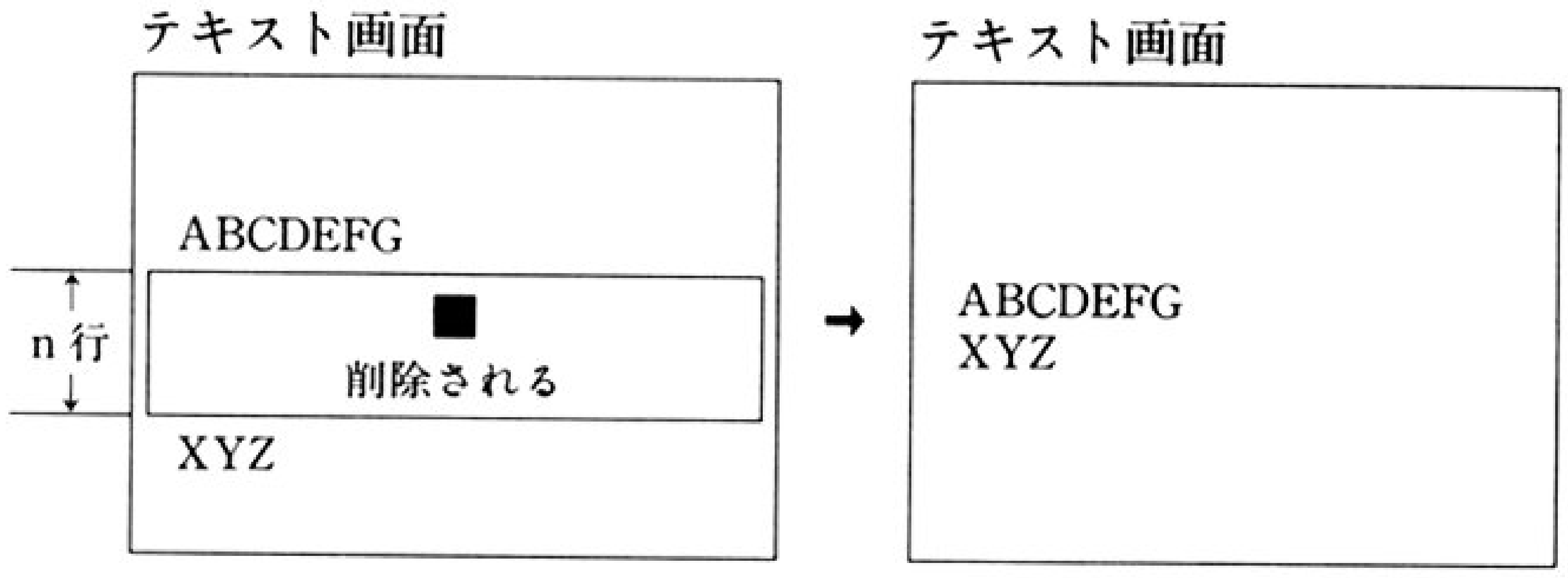
書式

```
#include <doslib.h>

int C_DEL (n) ;
int n;  /*  削除する行数  */
```

機能

カーソルのある行からn行削除します。
nに0を指定した場合、1とみなされます。
このファンクションコールは、CONデバイスによりサポートされます。



戻り値

処理が正常に終了したときは、0を返します。

参照関数

C_INS

C_DOWN

レベル 1

書　式

```
#include <doslib.h>

int C_DOWN (n) ;
int n; /*  カーソル位置の移動行数  */
```

機　能

カーソル位置をn行下へ移動します。
ただし、最終行より下へ移動しようとしても、スクロールアップは行われません。
nに0を指定した場合、1とみなされます。
このファンクションコールは、CONデバイスによりサポートされます。

戻り値

処理が正常に終了したときは、0を返します。

参照関数

C_UP

C_DOWN_S

レベル 1

書式

```
#include <doslib.h>

int C_DOWN_S ( ) ;
```

機能

カーソル位置を1行下へ移動します。
最終行より下へ移動するときには、スクロールアップが行われます。
このファンクションコールは、CON デバイスによりサポートされます。

戻り値

処理が正常に終了したときは、0を返します。

参照関数

C_UP_S

C_ERA_AL

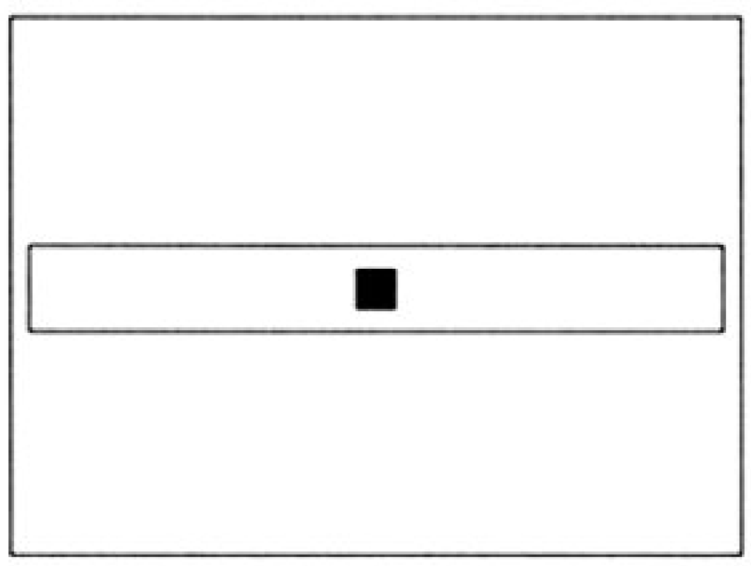
レベル 1

書　　式　# include <doslib. h>

int C_ERA_AL ( ) ;

機　　能　テキスト画面のカーソルのある行の左端から、右端までをクリアします。
このファンクションコールは、CON デバイスによりサポートされます。

テキスト画面



□の部分をクリアします。

戻 り 値　処理が正常に終了したときは、0 を返します。

参照関数　C_ERA_ED, C_ERA_ST

C_ERA_ED

レベル 1

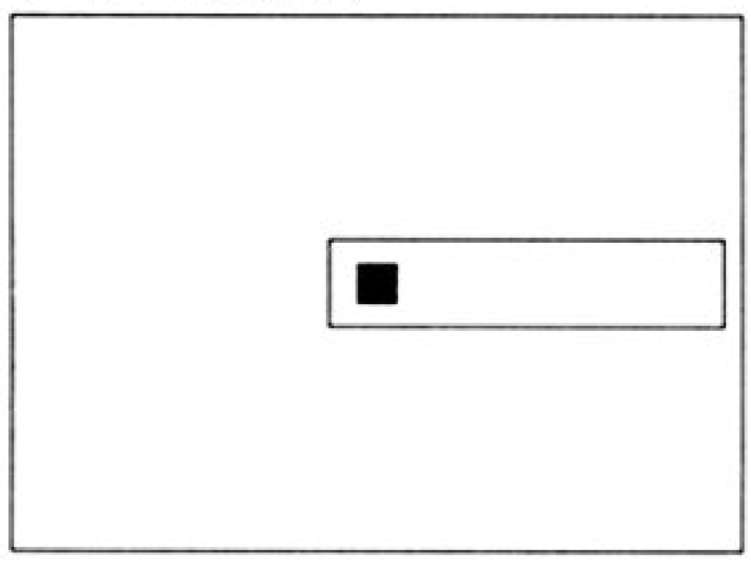
書　　式　# include <doslib. h>

int C_ERA_ED ( ) ;

機　　能　テキスト画面のカーソル位置から、カーソルがある行の右端までをクリアします。

このファンクションコールは、CON デバイスによりサポートされます。

テキスト画面



□の部分をクリアします。

戻 り 値　処理が正常に終了したときは、0 を返します。

参照関数　C_ERA_ST, C_ERA_AL

C_ERA_ST

レベル 1

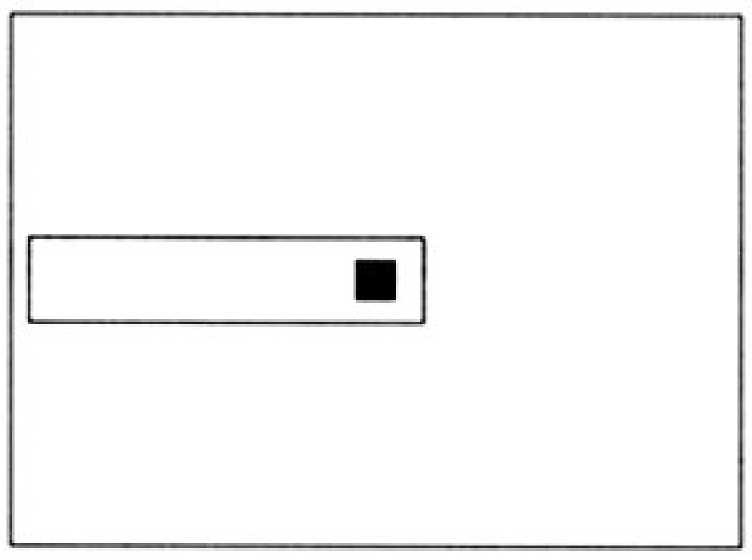
書　式　#include 〈doslib. h〉

int C_ERA_ST ( ) ;

機　能　テキスト画面のカーソルのある行の左端から、カーソル位置までをクリアします。

このファンクションコールは、CON デバイスによりサポートされます。

テキスト画面



□の部分をクリアします。

戻り値　処理が正常に終了したときは、0を返します。

参照関数　C_ERA_ED, C_ERA_AL

C_FNKMOD

レベル 1

書　式

```
#include <doslib.h>

int C_FNKMOD (n) ;
int n; /* ファンクションキー行のモード */
```

機　能

ファンクションキー行のモードを指定します。
スクロール範囲はリセットされます。
このファンクションコールは、CONデバイスによりサポートされます。
なお、nで指定できるモードは次の通りです。

0 ： ファンクションキー表示（スクロール範囲は0から31行分）
1 ： シフトファンクションキー表示（スクロール範囲は0から31行分）
2 ： 何も表示しない（スクロール範囲は0から31行分）
3 ： ふつうの行とする（スクロール範囲は0から32行分）
−1 ： 現在のモードを返す

戻り値

nに−1を指定したときのみ、現在のモード（0～3）を返します。

CHANGE_PR

レベル１

書　　式　# include <doslib. h>

```
void CHANGE_PR(   );
```

機　　能　バックグランドタスクの自分の実行権を放棄します。
次のタスクに切り替わります。

戻 り 値　戻り値はありません。

参照関数　OPEN_PR, KILL_PR, GET_PR, SUSPEND_PR, SLEEP_PR, SEND_PR, TIME_PR

CHDIR

レベル 1

書　式

```
#include <doslib.h>

int CHDIR (file);
unsigned char *file;  /*  ディレクトリ名格納領域へのポインタ  */
```

機　能　カレントディレクトリを、fileで指定したディレクトリに変更します。

戻り値　正の数（0も）で正常終了、負の数（−1以外も）でエラー。
通常は0が返りますが、特殊デバイスドライバが対象の時は、0以外の正の数の場合があります。

参照関数　MKDIR, RMDIR

CHGDRV

レベル 1

書　式

#include <doslib. h>

int CHGDRV (drive) ;
int drive ; /* ドライブ番号 */

機　能

カレントドライブを指定します。
指定するドライブ番号とドライブの対応関係は次のようになります。

0 ： ドライブA
1 ： ドライブB
2 ： ドライブC
3 ： ドライブD
⋮
⋮

戻り値

エラーならば指定した値以下の値を返します。
正常終了ならば指定した値より大きい値を返します。

プログラム例

```
#include        <doslib.h>

main()
{
        int     drive;          /* ドライブ番号 */

        drive = 1;
        printf("指定可能なドライブ数: %d¥n", CHGDRV(drive));

        if (CURDRV() < drive)
                puts("ERROR");
}
```

CHMOD

レベル 1

書　　式

```
#include <doslib.h>

int CHMOD (file, atr) ;
unsigned char *file; /* ファイル名格納領域へのポインタ */
int atr; /* ファイル属性 */
```

機　　能

fileで指定するファイルの属性を変更します。
atrに指定できる属性は次の通りです。
ただし、atrに−1を指定すると、属性の読み出しのみを行います。

ビット0	：	読み込み専用	組み合わせて0x00〜0x07まで指定できます。
ビット1	：	隠しファイル	
ビット2	：	システムファイル	
ビット3	：	ボリュームID	この3つのうちどれか1つを指定して上記ビット0〜ビット2と組み合わせた値を指定します。
ビット4	：	ディレクトリ	
ビット5	：	通常ファイル	

戻 り 値

負の数（−1以外も）でエラーとなります。
属性の変更に成功すると正の数（0も）を返します。
属性の読み出しに成功すると読み出した属性を返します。

C_INS

レベル 1

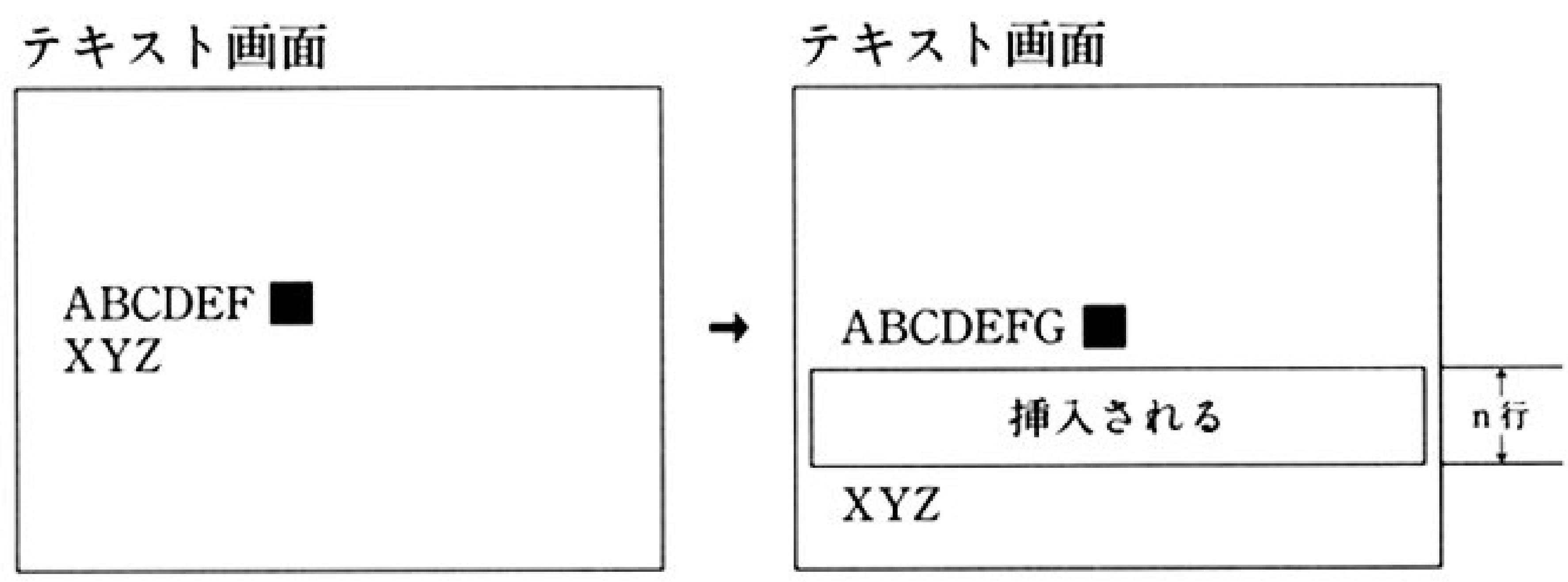
書式

```
#include <doslib.h>

int C_INS (n) ;
int n;  /*  挿入する行数  */
```

機能

カーソルのある行の直前にn行挿入します。
nに0を指定した場合、1とみなされます。
このファンクションコールは、CONデバイスによりサポートされます。



戻り値

処理が正常に終了したときは、0を返します。

参照関数

C_DEL

CINSNS

レベル 1

書　式	#include <doslib.h> int CINSNS ( ) ;
機　能	RS-232C 回線からの入力可・不可を調べます。
戻り値	入力不可のときには0、入力可能ならば0以外を返します。
参照関数	COUTSNS

CIRCLE

レベル 1

書　式

```
#include <iocslib. h>

int CIRCLE (circleptr) ;
struct CIRCLEPTR   {
  short x ;  /*  中心 X 座標  */
  short y ;  /*  中心 Y 座標  */
  unsigned short radius ;  /*  半径  */
  unsigned short color ;  /*  パレットコード  */
  short start ;  /*  円弧開始角度  */
  short end ;  /*  円弧終了角度  */
  unisgned short ratio ;  /*  比率  */
} *circleptr ;
```

機　能

グラフィック画面にサークルを描画します。
円弧開始角度、円弧終了角度は符号つき2バイト整数であり、0～360の角度で円弧を描きます。
範囲外の角度を指定した場合は、360を指定した事とみなされます。
負の値を指定すると扇型を描きます（角度は値の絶対値となります）。
比率は符号なし2バイト整数であり、0～65535の値をとります。
比率が0～255のとき半径は横方向の半径を意味し、縦方向との比率はX:256となります。
比率が256～65535のとき半径は縦方向の半径を意味し、横方向との比率はX:256となります。

戻り値

終了コードを返します。

0 ：正常終了
−1 ：グラフィックは使用不可

C_LEFT

レベル 1

書　式

```
#include <doslib.h>

int C_LEFT (n) ;
int n; /* カーソル位置の移動桁数 */
```

機　能

カーソル位置をn桁左へ移動します。
ただし、最左端より左へ移動しようとしても、右スクロールは行われません。
nに0を指定した場合、1とみなされます。
このファンクションコールは、CONデバイスによりサポートされます。

戻り値

処理が正常に終了したときは、0を返します。

参照関数

C_RIGHT

CLIPPUT

レベル 0

書　式

```
#include <iocslib.h>

void CLIPPUT (XDOT, YDOT, ADDRESS, CLIPPTR) ;
int XDOT ;  /*  X ドット座標  */
int YDOT ;  /*  Y ドット座標  */
struct FNTBUF {
  short xl ;  /*  書き込むパターンの X 方向のドット数  */
  short yl ;  /*  書き込むパターンの Y 方向のドット数  */
  unsigned char buffer [] ;  /*  パターンデータ  */
} *ADDRESS ;
struct CLIPXY {
  short xs ;  /*  表示領域の先頭 X 座標  */
  short ys ;  /*  表示領域の先頭 Y 座標  */
  short xe ;  /*  表示領域の最終 X 座標+1  */
  short ye ;  /*  表示領域の最終 Y 座標+1  */
} *CLIPPTR ;
```

機　能

ADDRESSで指定されたアドレスから、XDOT および YDOT で指定されたドット座標へのパターンを書き出すとともに、CLIPPTR が示すクリップ座標に従いクリッピング処理を行います。

ADDRESS にはスーパーバイザ領域のアドレスも指定できますが、誤ったアドレスを指定しないように注意してください。

ADDRESS には、書き出すパターンの X 方向のドット数、Y 方向のドット数、およびパターンデータを格納してください。

buffer には、指定した X, Y 方向のドット数に必要なだけのデータを格納しておいてください。

構造体 FNTBUF には、データバッファの宣言のみで実体は存在していません。メモリ確保用の関数 malloc などで確保した領域を使用してください。

また CLIPPTR には、表示領域の先頭 X 座標、先頭 Y 座標、最終 X 座標+1、最終 Y 座標+1 を格納します。

戻り値

戻り値はありません。

C_LOCATE

レベル 1

書　式

```
#include <doslib.h>

int C_LOCATE (x, y) ;
int x ;
int y ;
```

機　能

カーソルを (x, y) で指定した位置に設定します。
このファンクションコールは、CON デバイスによりサポートされます。

戻り値

処理が正常に終了したときは、0 を返します。

CLOSE

レベル 1

書　式

```
#include <doslib.h>

int CLOSE (fileno) ;
int fileno ;  /*  ファイルハンドル  */
```

機　能

filenoで指定するファイルハンドルをクローズします。

戻り値

正の数（0も）で正常終了、負の数（−1以外も）でエラー。
通常は0が返りますが、特殊デバイスドライバが対象の時は、0以外の正の数の場合があります。

COMINP

レベル 1

書式	# include <doslib. h> int COMINP ( ) ;
機能	RS-232C 回線から 1 バイト入力します。 また、このときブレークチェックも行います（^C)。
戻り値	入力データを返します。
参照関数	COMOUT, CINSNS, INP232C

COMMON_CK

レベル 1

書　　式　# include <doslib. h>

int COMMON_CK (name) ;
unsigned char ＊name ;　/＊　ブロックの名前　＊/

機　　能　common 領域において、指定するブロックが存在するかどうかチェックします。
name には、チェックしたいブロックの名前を指定します。

戻 り 値　指定したブロックが存在する場合は、そのブロックの大きさをバイト数で返します。
存在しない場合は、エラーになり負の数を返します。

参照関数　COMMON_RD, COMMON_WT, COMMON_LK, COMMON_FRE, COMMON_DEL

プログラム例

```
#include        <stdio.h>
#include        <stdlib.h>
#include        <doslib.h>

main(int argc, char *argv[])
{
        char    *name;
        int     size;

        if (argc == 2)
                name = argv[1];
        else {
                puts("COMMON領域の名前を指定してください。");
                exit(1);
        }

        printf("COMMON領域%s", name);
        if ((size = COMMON_CK(name)) < 0)
                printf("は存在しません。¥n");
        else {
                printf("は存在します。¥n");
                printf("領域の大きさは %dバイト¥n", size);
        }
        exit(0);
}
```

COMMON_DEL

レベル 1

書　式

include <doslib. h>

int COMMON_DEL (name) ;
unsigned char *name ; /* ブロックの名前 */

機　能

common 領域において、指定されたブロックを消去します。
name は消去したいブロックの名前を指定します。

戻り値

負の数ならエラーを表します。

参照関数

COMMON_CK, COMMON_RD, COMMON_WT, COMMON_LK, COMMON_FRE

プログラム例

```
#include        <stdio.h>
#include        <stdlib.h>
#include        <doslib.h>

main(int argc, char *argv[])
{
        char    *name;

        if (argc == 2)
                name = argv[1];
        else {
                puts("COMMON領域の名前を指定してください。");
                exit(1);
        }

        printf("COMMON領域%s", name);
        if (COMMON_DEL(name) >= 0)
                printf("を削除しました。¥n");
        else
                printf("の削除に失敗しました。¥n");
        exit(0);
}
```

COMMON_FRE

レベル 1

書　式

```
# include <doslib. h>

int COMMON_FRE (name, pos, id_psp, len) ;
unsigned char *name; /* ブロックの名前 */
int pos; /* ロック解除するデータの先頭を指す */
int id_psp; /* ロック解除するときのID */
int len; /* ロック解除するデータの長さ */
```

機　能

common 領域において、指定されたブロック中のデータをロック解除します。ロック解除すると、id_psp で指定されたプロセス以外からでもアクセスできるようになります。
name にはブロックの名前を指定します。
pos にはロックしたときに指定した位置を指定します。
id_psp はロックしたときに付けたプロセス ID です。
また len はロックしたときに指定したバイト数です。
COMMON_FRE で指定するパラメータは、COMMON_LK で指定したパラメータと完全に同一でなければなりません。これは、ロックしたプロセス以外のプロセスがロック解除を不可能にするためです。

戻り値

負の数ならエラーを表します。

参照関数

COMMON_CK, COMMON_RD, COMMON_WT, COMMON_LK, COMMON_DEL

プログラム例

```
#include        <stdio.h>
#include        <stdlib.h>
#include        <doslib.h>

/* extern int    _PSP; */


main(int argc, char *argv[])
{
        char    *name;
        int     size;

        if (argc == 2)
                name = argv[1];
        else {
                puts("COMMON領域の名前を指定してください。");
                exit(1);
        }

        printf("COMMON領域 %s", name);
        if ((size = COMMON_CK(name)) < 0)
                printf("は存在しません。¥n");
        else if (COMMON_FRE(name, 0, _PSP, size) >= 0)
                printf("を%dでロック解除しました。¥n", _PSP);
        else
                printf("のロック解除に失敗しました。¥n");
        exit(0);
}
```

COMMON_LK

レベル 1

書　式

```
# include <doslib. h>

int COMMON_LK (name, pos, id_psp, len) ;
unsigned char *name ; /*  ブロックの名前 */
int pos ; /*  ロックするデータの先頭を指す */
int id_psp ; /*ロックするときのID */
int len ; /*  ロックするときのデータの長さ */
```

機　能

common 領域において、指定されたブロック中のデータをロックします。
ロックすると、id_psp で指定されたプロセス以外からアクセスできなくなります。
name にはブロックの名前を指定します。
pos にはブロック中のデータのロックする位置を指定します。
id_psp にはロックするプロセスのプロセス ID を指定します。
通常は自分自身の PSP を指定します。これはシステムで定義された変数_PSP にセットされています。
また len はロックするデータのバイト数です。
id_psp には適当な値を指定しないでください。さもないと、自分自身も common 領域をアクセスできなくなります。必ず自分自身の PSP アドレスを指定してください。
また、ロック解除は、自分自身以外のプロセスでは不可能ですので、ロックを解除せずにプログラムを終了すると、二度とロック解除できなくなりますので注意してください。

戻り値

負の数ならエラーを表します。

参照関数

COMMON_CK, COMMON_RD, COMMON_WT, COMMON_FRE, COMMON_DEL

プログラム例

```
#include        <stdio.h>
#include        <stdlib.h>
#include        <doslib.h>

/* extern int   _PSP; */


main(int argc, char *argv[])
{
        char    *name;
        int     size;

        if (argc == 2)
                name = argv[1];
        else {
                puts("COMMON領域の名前を指定してください。");
                exit(1);
        }

        printf("COMMON領域 %s", name);
        if ((size = COMMON_CK(name)) < 0)
                printf("は存在しません。¥n");
        else if (COMMON_LK(name, 0, _PSP, size) >= 0)
                printf("を%dでロックしました。¥n", _PSP);
        else
                printf("のロックに失敗しました。¥n");
        exit(0);
}
```

COMMON_RD

レベル1

書　式

```
# include <doslib.h>

int COMMON_RD (name, pos, buffer, len) ;
unsigned char *name; /* ブロックの名前 */
int pos; /* 読み始める位置 */
unsigned char * buffer; /* 読み出したデータを格納する領域 */
int len; /* 読み出すデータの長さ */
```

機　能

common領域において、指定されたブロックよりデータを読み出します。
nameにはブロックの名前を指定します。
posにはブロックに格納されているデータを、何バイト目から読み出すか指定します。
bufferには読み出したデータを格納する領域へのポインタを指定します。
またlenは読み出すバイト数です。

戻り値

実際に読み込んだバイト数を返します。
負の数ならエラーを表します。

参照関数

COMMON_CK, COMMON_WT, COMMON_LK, COMMON_FRE, COMMON_DEL

プログラム例

```
#include        <stdio.h>
#include        <stdlib.h>
#include        <doslib.h>

#define LEN     256


main(int argc, char *argv[])
{
        char    buf[256];
        char    *name;
        int     size;

        if (argc == 2)
                name = argv[1];
        else {
                puts("COMMON領域の名前を指定してください。");
                exit(1);
        }

        printf("COMMON領域%s", name);
        if (COMMON_CK(name) < 0)
                printf("は存在しません。¥n");
        else if ((size = COMMON_RD(name, 0, buf, LEN)) < 0)
                printf("の読み込みに失敗しました。¥n");
        else
                printf("から%dバイト読み込みました。¥n", size);
        exit(0);
}
```

COMMON_WT

レベル 1

書　　式

```
# include <doslib. h>

int COMMON_WT (name, pos, buffer, len) ;
unsigned char *name ; /* ブロックの名前 */
int pos ; /* 書き始める位置 */
unsigned char * buffer ; /* 書き込むデータの格納領域 */
int len ; /* 書き込むデータの長さ */
```

機　　能

common 領域において、指定されたブロックへデータを書き込みます。
name はブロックの名前です。
pos はブロック中のデータ領域に書き込み始める位置を指定します。
buffer は、書き込むデータを格納してある領域へのポインタです。
また len は書き込むバイト数です。
なお、len を 0 にすると、ブロックを切り詰めます。

戻 り 値

実際に書き込んだバイト数を返します。
負の数ならエラーを表します。

参照関数

COMMON_CK, COMMON_RD, COMMON_LK, COMMON_FRE, COMMON_DEL

プログラム例

```
#include        <stdio.h>
#include        <stdlib.h>
#include        <doslib.h>

#define LEN     6

static char     buf[] = "X68000";


main(int argc, char *argv[])
{
        char    *name;
        int     n;

        if (argc == 2)
                name = argv[1];
        else {
                puts("COMMON領域の名前を指定してください。");
                exit(1);
        }

        printf("COMMON領域%s", name);
        if (COMMON_CK(name) >= 0) {
                if ((n = COMMON_WT(name, 0, buf, LEN)) >= 0)
                        printf("へ%dbyte書き込みました。¥n", n);
                else
                        printf("への書込みに失敗しました。¥n");
        } else {
                if (COMMON_WT(name, 0, buf, LEN) >= 0)
                        printf("を新たに作成しました。¥n");
                else
                        printf("の新規作成に失敗しました。¥n");
        }
        exit(0);
}
```

COMOUT

レベル 1

書　　式

```
#include <doslib.h>

void COMOUT (code) ;
int code; /*  出力する文字  */
```

機　　能

RS-232C 回線へ１バイト出力します。
また、このときブレークチェックも行います（^C)。

戻 り 値

戻り値はありません。

参照関数

COMIN, COUTSNS, OUT232C

CONSNS

レベル 1

書　式

```
#include <doslib.h>

int CONSNS ( ) ;
```

機　能　画面への出力の可・不可を調べます。

戻り値　出力不可のときには0、出力可能ならば0以外を返します。

CONTRAST

レベル 1

書　式

```
#include <iocslib.h>

int CONTRAST (MODE) ;
int MODE; /*  コントラストのモード  */
```

機　能

コントラストの設定を行います。
MODEの設定値とその内容は次の通りです。

0〜15 ： 各コントラストの設定
−1 ： コントラストの通知要求をします。
−2 ： コントラストをシステムの既定値に設定します。

戻り値

設定変更前のコントラストを返します。

COUTSNS

レベル 1

書　　式	# include <doslib. h> int COUTSNS ( ) ;
機　　能	RS-232C 回線への出力の可・不可を調べます。
戻 り 値	出力不可のときには 0、出力可能ならば 0 以外を返します。
参照関数	CINSNS

C_PRINT

レベル 1

書　式

```
#include <doslib.h>

int C_PRINT (msgptr);
unsigned char *msgptr;  /* 文字列格納領域へのポインタ  */
```

機　能

ヌル文字で終了する文字列データを表示します。
このファンクションコールは、CONデバイスによりサポートされます。

戻り値

処理が正常に終了したときは、0を返します。

参照関数

C_PUTC

C_PUTC

レベル 1

書式

```
#include <doslib.h>

int C_PUTC (code);
int code;  /*  表示するデータ  */
```

機能

指定した1バイトデータを表示します。
このファンクションコールは、CONデバイスによりサポートされます。

戻り値

処理が正常に終了したときは、0を返します。

参照関数

C_PRINT

CREATE

レベル 1

書　　式

```
#include <doslib.h>

int CREATE (file, atr) ;
unsigned char *file;  /*  ファイル名格納領域へのポインタ  */
int atr;  /*  ファイル属性  */
```

機　　能

ファイルを新規に作成します。
atr には、次のファイル属性を指定します。

ビット0	読み込み専用	組み合わせで0x00～0x07まで指定できます。
ビット1	隠しファイル	
ビット2	システムファイル	
ビット3	ボリュームID	この3つのうちどれか1つを指定して上記ビット0～ビット2と組み合わせた値を指定します。
ビット4	ディレクトリ	
ビット5	通常のファイル	

戻 り 値

ファイルハンドルの値を返します。
負の数なら、エラーを表します。

C_RIGHT

レベル 1

書式 # include <doslib. h>

```
int C_RIGHT (n) ;
int n;  /*  カーソル位置の移動桁数  */
```

機能 カーソル位置をn桁右へ移動します。
ただし、最右端より右へ移動しようとしても、左スクロールは行われません。
nに0を指定した場合、1とみなされます。
このファンクションコールは、CON デバイスによりサポートされます。

戻り値 処理が正常に終了したときは、0を返します。

参照関数 C_LEFT

CRTCRAS

レベル 0

書　式

```
#include <iocslib.h>

int CRTCRAS (ADDRESS, LUSTER) ;
unsigned char *ADDRESS; /* 割り込み処理アドレス */
int LUSTER; /* ラスタ */
```

機　能

CRTC に設定した割り込みラスタになったときに割り込みを行います。
ADDRESS には割り込み処理アドレスを指定します（0 ならば割り込み禁止）。
LUSTER にはラスタを指定します。

戻り値

割り込み可能なときは 0、すでに使用中の場合には 0 以外の値を返します。

CRTMOD

レベル0

書　式

```
#include <iocslib.h>

int CRTMOD (CRTMODE) ;
int CRTMODE;  /*  CRTのモード  */
```

機　能

CRTのモードを指定します。
このときテキスト画面はクリアされて、表示モードとなり、テキストパレットは標準に戻ります。
グラフィック画面、およびスプライト・BGはクリアされず、無表示モードとなります。
CRTMODEに−1を指定すると、現在のCRT モードを返します。
CRTMODEが0x100以上ならば、下位8ビットでモードの切り替えのみ行います（画面クリア、パレット・コントラスト・表示モードの初期化はしません)。
CRTMODEに指定できる値とその内容は次の通りです。

CRTMODE= ディスプレイ解像度	表示ドット数 横×縦	テキスト色／ グラフィック色	グラフィック 横ドット数
0 = high	512×512	16/16	1024
1 = low	512×512	16/16	1024
2 = high	256×256	16/16	1024
3 = low	256×256	16/16	1024
4 = high	512×512	16/16	512
5 = low	512×512	16/16	512
6 = high	256×256	16/16	512
7 = low	256×256	16/16	512
8 = high	512×512	16/256	512
9 = low	512×512	16/256	512
10 = high	256×256	16/256	512
11 = low	256×256	16/256	512
12 = high	512×512	16/65536	512
13 = low	512×512	16/65536	512
14 = high	256×256	16/65536	512
15 = low	256×256	16/65536	512
16 = high	768×512	16/16	1024

戻り値

CRTMODEが−1のとき、現在のCRTモードを返します。
これ以外の場合には戻り値はありません。

C_UP

レベル 1

書　　式

```
#include <doslib.h>

int C_UP (n) ;
int n;  /*  カーソル位置の移動行数  */
```

機　　能

カーソル位置をn行上へ移動します。
ただし、先頭行より上へ移動しようとしても、スクロールダウンは行われません。
nに0を指定した場合、1とみなされます。
このファンクションコールは、CONデバイスによりサポートされます。

戻 り 値

処理が正常に終了したときは、0を返します。

参照関数

C_DOWN

C_UP_S

レベル 1

書　　式	#include <doslib. h> int C_UP_S ( ) ;
機　　能	カーソル位置を1行上へ移動します。 先頭行より上へ移動するときにはスクロールダウンが行われます。 このファンクションコールは、CON デバイスによりサポートされます。
戻 り 値	処理が正常に終了したときは、0を返します。
参照関数	C_DOWN_S

CURDIR

レベル 1

書　式

```
#include <doslib.h>

int CURDIR (drive, buffer) ;
int drive ;  /* ドライブ番号 */
unsigned char * buffer ;  /* パス名格納領域へのポインタ */
```

機　能

driveで指定したドライブのカレントディレクトリをbufferが指す領域に設定します。

driveに指定するドライブ番号は次の通りです。

0：カレントドライブ
1：ドライブA
2：ドライブB
⋮
⋮
⋮

また、bufferには次のようにパス名が設定されます。

" "　　　　　　　* ルートディレクトリの場合
"TEST¥DOC"　* ディレクトリ名が¥TEST¥DOCの場合

戻り値

正の数（0も）で正常終了、負の数（−1以外も）でエラー。

CURDRV

レベル 1

書　式　#include <doslib. h>

int CURDRV ( ) ;

機　能　カレントドライブの番号を調べます。

戻り値　カレントドライブがどのドライブか返します。
0ならばドライブA、1ならばドライブBとなります。

プログラム例

```
#include        <stdio.h>
#include        <doslib.h>

main()
{
        int     drive;          /* ドライブ番号 */

        drive = CURDRV() + 'A';
        printf("カレントドライブは、%cドライブです。¥n", drive);
}
```

C_WIDTH

レベル 1

書　　式

```
#include <doslib.h>

int C_WIDTH (n) ;
int n; /*  画面のモード  */
```

機　　能

画面サイズの設定を行います。
このファンクションコールは、CON デバイスによりサポートされます。
なお、nで指定できるモードは次の通りです。

0：高解像度　768×512　グラフィックなし
1：高解像度　768×512　グラフィック 16 色
2：高解像度　512×512　グラフィックなし
3：高解像度　512×512　グラフィック 16 色
4：高解像度　512×512　グラフィック 256 色
5：高解像度　512×512　グラフィック 65536 色
−1：現在のモードを返す

戻 り 値

設定変更前のモード（0〜5）を返します。

C_WINDOW

レベル 1

書　式

```
#include <doslib.h>

int C_WINDOW (ys, yl) ;
int ys; /* 始点Y座標 */
int yl; /* スクロール行数 */
```

機　能

スクロールの範囲を設定します。
このとき、カーソルはホームポジションに移動します。
ys+yl の値は、ファンクションキー行のモードが 3 のときは 32 まで、これ以外のときは 31 までとします。
絶対座標の (0, ys) が論理座標の (0, 0) となります。
なお、このファンクションコールは、CON デバイスによりサポートされます。

戻り値

処理が正常に終了したときは、0 を返します。

DAKJOB

レベル０

書　式

include <iocslib. h>

int DAKJOB (BUFEND) ;
unsigned char ＊BUFEND ; /＊ ヌル文字のアドレス ＊/

機　能

濁点処理を行います。
BUFEND には、全角文字列の終端であるヌル文字のアドレスを指定します。
最後の全角文字に濁点処理をできないときは、文字列に（゛）を追加しますので、ヌル文字の後に３バイト分の領域が必要です。また、（゛）を追加した場合は（゛）の後にヌル文字を付けますので、文字列の途中を指定すると（゛）の後の文字列が切れてしまいます。したがって文字列の途中を指定しないでください。

戻り値

次の内容を返します。

0 ： 最後の全角文字に濁点処理をした
2 ： （゛）を追加した

参照関数

HANJOB

プログラム例

```
#include        <stdio.h>
#include        <iocslib.h>

static char     str[13] = "かきくけこ"; /* 全角文字列バッファ */

main()
{
        char    *endp;          /* バッファ最終アドレス */

        printf("%s¥n", str);
        endp = str + strlen(str);

        DAKJOB(endp);

        printf("%s¥n", str);
}
```

DATEASC

レベル 0

書式

```
#include <iocslib.h>

int DATEASC (BINDATE, BUF) ;
int BINDATE; /* 日付 (2進数) */
unsigned char *BUF; /* 文字列格納領域へのポインタ */
```

機能

2進データを、日付を表す文字列に変換し、格納します。
BINDATEの形式と指定内容は次の通りです。

FFFFyyyy yyyyyyyy mmmmmmmm dddddddd

FFFF	：	文字列の変換形式
		0:1987/08/07
		1:1987－08－07
		2:87/08/07
		3:87－08－07
yyyyyyyyyyyy	：	年 (1980～2079)
mmmmmmmm	：	月 (01～12)
dddddddd	：	日 (01～31)

BUFには、日付を表す文字列の格納領域の先頭アドレスを指定します。
この領域は最低11バイト以上確保してください。
BUFにはスーパーバイザ領域のアドレスも指定できますが、誤ったアドレスを指定しないよう注意してください。

戻り値

エラーが発生したときは－1を返します。

プログラム例

```
#include        <stdio.h>
#include        <iocslib.h>

#define STYLE           0
#define YEAR            1990
#define MONTH           3
#define DATE            14

main()
{
        int     bin;            /* 日付(2進数) */
        char    buf[11];        /* 文字列格納領域 */

        bin = STYLE << 28;
        bin |= YEAR << 16;
        bin |= MONTH << 8;
        bin |= DATE << 0;
        DATEASC(bin, buf);
        printf("date = %s¥n", buf);
}
```

DATEBIN

レベル 0

書式

```
#include <iocslib.h>

int DATEBIN (BCDDATE) ;
int BCDDATE; /* 日付 */
```

機能

日付を BCD から 2 進数に変換します。
BCDDATE の形式と指定内容は次の通りです。

0000WWWW YYYYYYYY MMMMMMMM DDDDDDDD

WWWW	：	曜日カウンタ (0～6)　(0：日曜日)
YYYYYYYY	：	年 (00～99。1980 年からの相対年数)
MMMMMMMM	：	月 (01～12)
DDDDDDDD	：	日 (01～31)

上記の年，月，日はいずれも BCD2 桁です。

戻り値

2 進数に変換した日付を返します。日付の形式は次の通りです。

wwwwyyyy yyyyyyyy mmmmmmmm dddddddd

wwww	：	曜日 (0～6)
yyyyyyyyyyyy	：	年 (1980～2079)
mmmmmmmm	：	月 (01～12)
dddddddd	：	日 (01～31)

プログラム例

```
#include        <stdio.h>
#include        <iocslib.h>

main()
{
        int     bcd;    /* 日付(BCD) */
        int     bin;    /* 日付(2進数) */

        bcd = BINDATEGET();
        bin = DATEBIN(bcd);

        printf("day   = %d¥n", (bin >> 28) & 0x0f);
        printf("year  = %d¥n", (bin >> 16) & 0x0fff);
        printf("month = %d¥n", (bin >> 8) & 0xff);
        printf("date  = %d¥n", (bin >> 0) & 0xff);
}
```

DATECNV

レベル０

書　　式

include <iocslib. h>

int DATECNV (ADDRESS) ;
unsighed char ＊ ADDRESS ; /＊ 日付を表す文字列へのポインタ ＊/

機　　能

日付を表す文字列を２進数に変換します。
ADDRESSには、日付を表す文字列の先頭アドレスを指定します。
ADDRESSにはスーパーバイザ領域のアドレスも指定できますが、誤ったアドレスを指定しないよう注意してください。
また、日付データは次のように指定します。

"1987/08/07"

区切り記号は/（スラッシュ）でも－（マイナス）でも、それら以外でもかまいません。

戻 り 値

エラーが発生したときは－1を返します。
ただし、うるう年や大小の月の判定は行いません。
変換を行ったときは２進数に変換した日付を返します。
日付の形式は次の通りです。

0000yyyy yyyyyyyy mmmmmmmm dddddddd

yyyyyyyyyyyy ＝ 年（1980～2079）
mmmmmmmm ＝ 月（01～12）
dddddddd ＝ 日（01～31）

プログラム例

```
#include        <stdio.h>
#include        <iocslib.h>

main()
{
        int     bin;            /* 日付(2進数) */

        bin = DATECNV("1990/03/14");

        printf("year  = %d¥n", (bin >> 16) & 0x0fff);
        printf("month = %d¥n", (bin >> 8) & 0xff);
        printf("date  = %d¥n", (bin >> 0) & 0xff);
}
```

DAYASC

レベル 0

書　式

include <iocslib. h>

void DAYASC (BINDAY, BUF) ;
int BINDAY ; /＊ 曜日 ＊/
unsigned char ＊ BUF ; /＊ 文字列格納領域へのポインタ ＊/

機　能

2進数を、曜日を表す文字列に変換し、格納します。
BINDAY には曜日 (0～6) を指定します。
範囲外の値を指定すると、8で割った余りが使用されます (つねに0～7になります)。
BUF には、曜日を表す文字列の格納領域の先頭アドレスを指定します。
この領域は最低3バイト以上確保してください。
BUF にはスーパーバイザ領域のアドレスも指定できますが、誤ったアドレスを指定しないように注意してください。

戻り値

戻り値はありません。
BUF には "日"、"月"、"火"、"水"、"木"、"金"、"土"、または "?" のいずれかがセットされます。
"?" がセットされたときは、7または8で割った余りが7になる値を指定したことを表します。

プログラム例

```
#include        <stdio.h>
#include        <iocslib.h>

main()
{
        char    buf[3];         /* 文字列格納領域 */
        int     y;              /* 曜日 */

        for (y = 0; y < 7; y++) {
                DAYASC(y, buf);
                printf("曜日 = %s曜日¥n", buf);
        }
}
```

DEFCHR

レベル 0

書　式　#include <iocslib. h>

int DEFCHR (TYPE, CODE, ADDRESS) ;
int TYPE ;　/＊　文字の大きさ　＊/
int CODE ;　/＊　漢字コード　＊/
unsigned char ＊ADDRESS ;　/＊　外字パターン格納領域のアドレス　＊/

機　能　CODE で指定された外字に、ADDRESS で指定した外字パターンを定義します。
TYPE には 8 または 12 を指定します。
TYPE に 0 を指定した場合は 8 と同じです。
CODE にはシフト JIS コードを指定しますが、JIS 漢字コード (0x7621 ～0x777E) を指定することもできます。
TYPE に 0 か 8 を指定した場合は 32 バイトの外字パターンが、TYPE に 12 を指定した場合は 72 バイトの外字パターンが必要です。
それぞれ、16×16 ドット、24×24 ドットのパターンになります。

戻り値　処理が正常に終了したときは 0、指定した漢字コードが外字でなかったときにはエラーコード (−1) を返します。

プログラム例

```
#include        <stdio.h>
#include        <iocslib.h>

main()
{
        char    buf[32];        /* 漢字パターン格納領域 */
        int     type;           /* 文字の大きさ */
        int     code;           /* 外字コード */
        int     i;

        for (i = 0; i < 32; i++)
                buf[i] = 0xaa;
        type = 8;
        code = 0xebe9;
        DEFCHR(type, code, buf);

        printf("0xebe9: %c%c¥n", 0xeb, 0xe9);
}
```

DELETE

レベル 1

書　　式

```
#include <doslib.h>

int DELETE (file);
unsigned char *file;  /*  削除するファイル名へのポインタ  */
```

機　　能

fileで指定するファイルを削除します。
ワイルドカードや、ディレクトリの指定はできません。
また、オープンされているファイルに対して実行した場合はエラーになります。

戻 り 値

正の数（0も）で正常終了、負の数（−1以外も）でエラー。
通常は0が返りますが、特殊デバイスドライバが対象の時は0以外の正の数の場合があります。

DENSNS

レベル 0

書式

```
#include <iocslib.h>

void DENSNS ( ) ;
```

機能

電卓処理をします。
電卓処理は、B_KEYINP、B_KEYSNS 等のキー入力関数で処理されます。
したがって、マウスのみで動作するプログラムのようにキー入力を行わない場合には、ソフトキーボードが表示されていても電卓が使えないことがあります。
このような場合、この DENSNS を呼びながらマウス処理をしておけば、電卓が使用できるようになります。

戻り値

戻り値はありません。

DISKRED

レベル1

書式

```
#include <doslib.h>

void DISKRED (adr, drive, sector, seclen) ;
unsigned char *adr;  /*  データ読み込み領域へのポインタ  */
int drive;  /*  ドライブ番号  */
int sector;  /*  読み込むセクタの先頭  */
int seclen;  /*  読み込むセクタ数  */
```

機能

ブロックデバイスの直接入力を行います。
読み込みはセクタ単位で行うので、adr から始まるバッファのサイズは読み込みを行うデバイスの1セクタあたりのバイト数の整数倍にしてください。
なお、標準のデバイスでは1セクタは1024バイトですので、1024の倍数を指定してください。

drive は、読み込みを行うドライブ番号です。
0がカレントドライブ、1がドライブA、2がドライブBとなります。
仮想ドライブや、仮想ディレクトリに割り当てた実ドライブはアクセスできません。

sector および seclen は、65535までを指定してください。これを超えるセクタ位置やセクタ数は指定できません。
もし大容量のドライブをアクセスしたいならば、DISKRED2関数を使用してください。

戻り値

戻り値はありません。

参照関数

DISKWRT, DISKRED2, DISKWRT2

プログラム例

```
#include        <stdio.h>
#include        <doslib.h>

#define SECT_SIZE       1024

main()
{
        unsigned char   buf[SECT_SIZE]; /* データ格納バッファ*/
        int     drive = 1;              /* ドライブ番号 */
        int     sector = 5;             /* 読込むセクタの先頭 */
        int     seclen = 1;             /* 読み込むセクタ数 */
        int     i;

        DISKRED(buf, drive, sector, seclen);

        for (i = 0; i < 128; i++) {
                printf("%02x", buf[i]);
                printf("%c", (((i % 16) == 15) ? '¥n' : ' '));
        }
}
```

DISKRED2

レベル 1

書式

```
# include <doslib. h>

void DISKRED2 (adr, drive, sector, seclen) ;
unsigned char *adr ; /* データ読み込み領域へのポインタ */
int drive ; /* ドライブ番号 */
long sector ; /* 読み込むセクタの先頭 */
long seclen ; /* 読み込むセクタ数 */
```

機能

ブロックデバイスの直接入力を行います。
読み込みはセクタ単位で行うので、adrから始まるバッファのサイズは読み込みを行うデバイスの1セクタあたりのバイト数の整数倍にしてください。
なお、標準のデバイスでは1セクタは1024バイトですので、1024の倍数を指定してください。

driveは、読み込みを行うドライブ番号です。
0がカレントドライブ、1がドライブA、2がドライブBとなります。
仮想ドライブや、仮想ディレクトリに割り当てた実ドライブはアクセスできません。

DISKRED関数に対して、DISKRED2関数は大容量ドライブ・特殊ドライブに対応しており、sectorやseclenの値を65535を超えるロングワードで指定することができます。

戻り値

戻り値はありません。

参照関数

DISKRED, DISKWRT, DISKWRT2

プログラム例

```
#include        <stdio.h>
#include        <doslib.h>

#define SECT_SIZE       1024

main()
{
        unsigned char   buf[SECT_SIZE]; /* データ格納バッファ*/
        int     drive = 1;              /* ドライブ番号 */
        long    sector = 5L;            /* 読込むセクタの先頭 */
        long    seclen = 1L;            /* 読み込むセクタ数 */
        int     i;

        DISKRED2(buf, drive, sector, seclen);
        for (i = 0; i < 128; i++) {
                printf("%02x", buf[i]);
                printf("%c", (((i % 16) == 15) ? '¥n' : ' '));
        }
}
```

DISKWRT

レベル 1

書　式

```
#include <doslib.h>

void DISKWRT (adr, drive, sector, seclen) ;
unsigned char *adr; /* データの先頭ポインタ */
int drive; /* ドライブ番号 */
int sector; /* 書き出しセクタの先頭 */
int seclen; /* 書き出しセクタ数 */
```

機　能

ブロックデバイスの直接出力を行います。
書き出しはセクタ単位で行うので、adr から始まるバッファのサイズは書き出しを行うデバイスの1セクタあたりのバイト数の整数倍にしてください。
なお、標準のデバイスでは1セクタは1024バイトですので、1024の倍数を指定してください。

drive は、書き出しを行うドライブ番号です。
0がカレントドライブ、1がドライブA、2がドライブBとなります。
仮想ドライブや、仮想ディレクトリに割り当てた実ドライブはアクセスできません。

sector および seclen は、65535までを指定してください。これを超えるセクタ位置やセクタ数は指定できません。
もし大容量のドライブをアクセスしたいならば、DISKWRT2 関数を使用してください。

戻り値

戻り値はありません。

参照関数

DISKRED, DISKRED2, DISKWRT2

プログラム例

```
#include        <stdio.h>
#include        <doslib.h>

#define SECT_SIZE       1024


/* 注意！ */ /* 注意！ */ /* 注意！ */
/* このプログラムを実行するとドライブBの内容を破壊します。 */
main()
{
        void    dump(char *);
        char    buf[SECT_SIZE]; /* データ格納バッファ */
        int     drive = 2;      /* ドライブ番号 */
        int     sector = 5;     /* 書き出しセクタの先頭 */
        int     seclen = 1;     /* 書き出しセクタ数 */
        int     i;


        DISKRED(buf, drive, sector, seclen);
        dump(buf);

        for (i = 0; i < SECT_SIZE; i++)
                buf[i] = 0x40;
        DISKWRT(buf, drive, sector, seclen);

        DISKRED(buf, drive, sector, seclen);
        dump(buf);
}


static void     dump(char *buf)
{
        int     i;

        for (i = 0; i < SECT_SIZE; i++) {
                printf("%02x", buf[i] & 0xff);
                printf("%c", (((i % 16) == 15) ? '¥n' : ' '));
        }
}
```

DISKWRT2

レベル 1

書　式

```
# include <doslib. h>

void DISKWRT2 (adr, drive, sector, seclen) ;
unsigned char *adr ; /* 書き出すデータの先頭ポインタ */
int drive ; /* ドライブ番号 */
long sector ; /* 書き出しセクタの先頭 */
long seclen ; /* 書き出しセクタ数 */
```

機　能

ブロックデバイスの直接出力を行います。
書き出しはセクタ単位で行うので、adrから始まるバッファのサイズは書き出しを行うデバイスの1セクタあたりのバイト数の整数倍にしてください。
なお、標準のデバイスでは1セクタは1024バイトですので、1024の倍数を指定してください。

driveは、書き出しを行うドライブ番号です。
0がカレントドライブ、1がドライブA、2がドライブBとなります。
仮想ドライブや、仮想ディレクトリに割り当てた実ドライブはアクセスできません。

DISKRED関数に対して、DISKRED2関数は大容量ドライブ・特殊ドライブに対応しており、sectorやseclenの値を65535を超えるロングワードで指定することができます。

戻り値

戻り値はありません。

参照関数

DISKWRT, DISKWRT, DISKRED2

プログラム例

```
#include        <stdio.h>
#include        <doslib.h>

#define SECT_SIZE       1024


/* 注意! */ /* 注意! */ /* 注意! */
/* このプログラムを実行するとドライブBの内容を破壊します。 */
main()
{
        void    dump(char *);
        char    buf[SECT_SIZE]; /* データ格納バッファ */
        int     drive = 2;      /* ドライブ番号 */
        long    sector = 5L;    /* 書き出しセクタの先頭 */
        long    seclen = 1L;    /* 書き出しセクタ数 */
        int     i;


        DISKRED2(buf, drive, sector, seclen);
        dump(buf);

        for (i = 0; i < SECT_SIZE; i++)
                buf[i] = 0x40;
        DISKWRT2(buf, drive, sector, seclen);

        DISKRED2(buf, drive, sector, seclen);
        dump(buf);
}


static void     dump(char *buf)
{
        int     i;

        for (i = 0; i < SECT_SIZE; i++) {
                printf("%02x", buf[i] & 0xff);
                printf("%c", (((i % 16) == 15) ? '¥n' : ' '));
        }
}
```

DMAMODE

レベル 0

書式　# include <iocslib, h>

int DMAMODE ( ) ;

機能　DMA の実行モードを調べます。

戻り値　DMA の実行モードを返します。

0x00　：何もしていない
0x8A　：転送中（DMAMOVE 実行中）
0x8B　：転送中（DMAMOV_L 実行中）
0x8C　：転送中（DMAMOV_A 実行中）

プログラム例

```
#include        <stdio.h>
#include        <iocslib.h>

#define DIRECT  0x00    /* buf1 → buf2 */
#define SRCINC  0x04    /* buf1インクリメント */
#define DSTINC  0x01    /* buf2インクリメント */
#define SIZE    128     /* 転送バイト数 */

main()
{
        unsigned char   buf1[SIZE];     /* 転送元 */
        unsigned char   buf2[SIZE];     /* 転送先 */
        int     mode;           /* モード */
        int     i;

        for (i = 0; i < SIZE; i++)
                buf1[i] = i;
        if (DMAMODE() == 0) {
                mode = DIRECT | SRCINC | DSTINC;
                DMAMOVE(buf1, buf2, mode, SIZE);
                for (i = 0; i < SIZE; i++) {
                        printf("%02x, ", buf2[i]);
                        if (i % 16 == 15)
                                putchar('¥n');
                }
        }
}
```

DMAMOV_A

レベル 0

書式

```
#include <iocslib.h>
void DMAMOV_A (TBLADDRESS, ADDRESS, MODE, TBLCNT) ;
struct CHAIN *TBLADDRESS; /* データチェーンテーブルのアドレス
                                                            */
unsigned char *ADDRESS; /* 転送先アドレス */
int MODE; /* モード */
int TBLCNT; /* データチェーンテーブルの個数 */
struct CHAIN {
  int adr; /* 先頭アドレス */
  unsigned short len; /* 長さ */
};
```

機能

アレイチェーンによるDMA転送を行います。
TBLADDRESSには転送元データチェーンテーブルのアドレスを指定します。
このアドレスには、先頭アドレス、長さ、先頭アドレス、長さ、・・・の順で格納します。
ADDRESSには転送先アドレスを指定します。
TBLCNTには、転送データチェーンテーブルの個数を指定します。
MODEのDIRによって、転送先と転送元を切り換えることができます。
MODEの形式と、各項目の指定内容は次の通りです。

7	6	5	4	3	2	1	0
DIR	0	0	0	MAC		DAC	

● DIR
- 0 : (TBLADDRESS) から ADDRESS へ
- 1 : ADDRESS から (TBLADDRESS) へ

● MAC (TBLADDRESS)/DAC (ADDRESS)
- 00 : カウントしない
- 01 : インクリメント
- 10 : デクリメント
- 11 : 指定不可

戻り値

戻り値はありません。

プログラム例

```
#include        <stdio.h>
#include        <iocslib.h>

#define DIR     0x00    /* tbl → dbuf */
#define MAC     0x04    /* tblインクリメント */
#define DAC     0x01    /* dbufインクリメント */
#define T_NUM   3       /* chainテーブル個数 */

main()
{
        struct CHAIN    tbl[T_NUM];     /* chainテーブル */
        char    sbuf[256];      /* 転送元 */
        char    dbuf[100];      /* 転送先 */
        int     mode;           /* モード */
        int     i;

        mode = DIR | MAC | DAC;
        for (i = 0; i < 255; i++)
                sbuf[i] = i;
        for (i = 0; i < T_NUM; i++) {
                tbl[i].adr = sbuf + i * 0x50;
                tbl[i].len = 0x20;
        }
        DMAMOV_A(tbl, dbuf, mode, T_NUM);
        for (i = 0; i < 96; i++) {
                printf("%02x", dbuf[i] & 0xff);
                printf("%c", (((i % 16) == 15) ? '¥n' : ' '));
        }
}
```

DMAMOVE

レベル 0

書式 #include <iocslib.h>

```
void DMAMOVE (ADDRESS1, ADDRESS2, MODE, BYTE) ;
unsigned char *ADDRESS1 ;  /* 転送元アドレス  */
unsigned char *ADDRESS2 ;  /* 転送先アドレス  */
int MODE ;  /*  モード  */
int BYTE ;  /*  転送するデータのバイト数  */
```

機能 DMA 転送を行います。

転送するデータのバイト数は 0xFF00 以内にしてください。

ADDRESS1 には転送元アドレス、ADDRESS2 には転送先アドレスを指定します。

MODE の DIR によって、ADDRESS1 を転送先アドレスに、ADDRESS2 を転送元アドレスに切り換えることができます。

MODE の形式と、各項目の指定内容は次の通りです。

7	6	5	4	3	2	1	0
DIR	0	0	0	MAC		DAC	

● DIR

0 ： ADDRESS1 から ADDRESS2 へ

1 ： ADDRESS2 から ADDRESS1 へ

● MAC (ADDRESS1)/DAC (ADDRESS2)

00 ： カウントしない

01 ： インクリメント

10 ： デクリメント

11 ： 指定不可

戻り値 戻り値はありません。

プログラム例

```
#include        <stdio.h>
#include        <iocslib.h>

#define DIRECT  0x00    /* sbuf → dbuf */
#define SRCINC  0x04    /* sbufインクリメント */
#define DSTINC  0x01    /* dbufインクリメント */
#define SIZE    128     /* 転送バイト数 */

main()
{
        unsigned char   sbuf[SIZE];     /* 転送元 */
        unsigned char   dbuf[SIZE];     /* 転送先 */
        int     mode;           /* モード */
        int     i;

        for (i = 0; i < SIZE; i++)
                sbuf[i] = i;
        mode = DIRECT | SRCINC | DSTINC;
        DMAMOVE(sbuf, dbuf, mode, SIZE);
        for (i = 0; i < SIZE; i++) {
                printf("%02x", dbuf[i]);
                printf("%c", (((i % 16) == 15) ? '¥n' : ' '));
        }
}
```

DMAMOV_L

レベル 0

書式

```
#include <iocslib.h>
void DMAMOV_L (TBLADDRESS, ADDRESS, MODE) ;
struct CHAIN2 *TBLADDRESS ; /* データチェーンテーブルのアドレス
                                                                   */
unsigned char *ADDRESS ; /* 転送先アドレス */
int MODE ; /* モード */
struct CHAIN2 {
  int adr ; /* 先頭アドレス */
  unsigned short len ; /* 長さ */
  struct CHAIN2 *next ; /*次のテーブルのアドレス */
} ;
```

機能

リンクアレイチェーンよるDMA転送を行います。

TBLADDRESSには、転送元データチェーンテーブルのアドレスを指定します。

このアドレスには、先頭アドレス・長さ・次のテーブルアドレスを格納し、次のテーブルにチェーンします。

最後のテーブルでは、次のテーブルアドレスに0を設定します。

ADDRESSには転送先アドレスを指定します。

MODEのDIRによって、転送先と転送元を切り換えることができます。

MODEの形式と、各項目の指定内容は次の通りです。

7	6	5	4	3	2	1	0
DIR	0	0	0	MAC		DAC	

● DIR

0 : (TBLADDRESS) から ADDRESS へ

1 : ADDRESS から (TBLADDRESS) へ

● MAC (TBLADDRESS)/DAC (ADDRESS)

00 : カウントしない

01 : インクリメント

10 : デクリメント

11 : 指定不可

戻り値

戻り値はありません。

プログラム例

```
#include        <stdio.h>
#include        <iocslib.h>

#define DIRECT  0x00    /* sbuf → dbuf */
#define SRCINC  0x04    /* sbufインクリメント */
#define DSTINC  0x01    /* dbufインクリメント */
#define SIZE    256

static char     sbuf[SIZE];
static char     dbuf[SIZE];

struct CHAIN2   tbl3 = {        /* テーブル3 */
        &sbuf[0x40],
        0x20,
        0                       /* つづきなし */
};
struct CHAIN2   tbl2 = {        /* テーブル2 */
        &sbuf[0x00],
        0x20,
        &tbl3                   /* テーブル3へつづく */
};
struct CHAIN2   tbl1 = {        /* テーブル1 */
        &sbuf[0x20],
        0x20,
        &tbl2                   /* テーブル2へつづく */
};

main()
{
        int     mode;           /* モード */
        int     i;

        for (i = 0; i < SIZE; i++)
                sbuf[i] = i;
        mode = DIRECT | SRCINC | DSTINC;

        DMAMOV_L(&tbl1, dbuf, mode);

        for (i = 0; i < 96; i++) {
                printf("%02x", dbuf[i] & 0xff);
                printf("%c", (((i % 16) == 15) ? '¥n' : ' '));
        }
}
```

DRVCTRL

レベル 1

書　式

```
#include <doslib.h>

int DRVCTRL (mode, drive) ;
int mode;  /*  オペレーションモード  */
int drive;  /*  ドライブ番号  */
```

機　能

指定ドライブの状態のセンスと設定を行います。
mode と drive の設定値は次の通りです。

● mode

0 ： ドライブの状態のセンス（D0 のビット 7～ビット 0）
1 ： イジェクトする（該当ドライブでオープンされているファイルがあった場合はイジェクトしない）。
2 ： イジェクトを禁止する（mode ＝ 1でイジェクトを指定してもこちらの指定が優先される）。
3 ： イジェクトを許可する（該当ドライブでオープンされているファイルはクローズされないが、バッファは自動的にフラッシュされる）。
4 ： ディスクがセットされていないとき、LED 点滅。
5 ： ディスクがセットされていないとき、LED 消灯。

● drive

0 ： カレントドライブ
1 ： A ドライブ
2 ： B ドライブ
⋮

戻り値

指定ドライブの状態を返します。各ビットの内容は次の通りです。

7	6	5	4	3	2	1	0
LED 点滅	EJECT 禁止	BUFFER 有り	ユーザー 使用禁止	PROTECT	NOTREADY	メディア 挿入	誤挿入

（いずれもビット ＝ 1のとき、表記の状態であることを示します）
なお、PROTECT（プロテクト ＝ 1）および NOTREADY（ノットレディ ＝ 1）は、mode に 0 を指定し、なおかつメディア挿入のときにのみ有効です。

DRVCTRL

レベル 1

プログラム例

```
#include        <doslib.h>

#define EJECT   1               /* イジェクト */
#define CRDRV   1               /* Aドライブ */

main()
{
        int     mode;           /* オペレーション・モード */
        int     drive;          /* ドライブ番号 */

        mode = EJECT;
        drive = CRDRV;
        DRVCTRL(mode, drive);
}
```

DRVXCHG

レベル 1

書式

#include <doslib. h>

void DRVXCHG (old, new) ;
int old ; /＊ 旧ドライブ番号 ＊/
int new ; /＊ 新ドライブ番号 ＊/

機能

old のドライブと new のドライブを入れ替えます。
old, new の値は次のように指定します。

0 ： カレントドライブ
1 ： ドライブA
2 ： ドライブB
⋮

戻り値

戻り値はありません。

プログラム例

```
#include        <doslib.h>

main()
{
        int     old;    /* 旧ドライブ番号 */
        int     new;    /* 新ドライブ番号 */

        old = 0;
        new = 1;
        DRVXCHG(old, new);
}
```

DSKFRE

レベル 1

書　式

```
#include <doslib.h>

int DSKFRE (drive, buffer) ;
int drive ;  /*  ドライブ番号  */
struct FREEINF  *buffer ;  /*  残り容量格納領域へのポインタ  */
```

機　能

ディスクの残り容量を調べ、その結果を buffer に格納します。
buffer の形式は次の通りです。

```
struct FREEINF  {
  unsigned short free ;  /*  使用可能なクラスタ数  */
  unsigned short max ;  /*  総クラスタ数  */
  unsigned short sec ;  /*  1クラスタあたりのセクタ数  */
  unsigned short byte ;  /*  1セクタあたりのバイト数  */
} ;
```

戻り値

使用可能なバイト数（最大2ギガバイト）を返します。
負の数ならエラーを表します。

プログラム例

```
#include        <stdio.h>
#include        <doslib.h>

#define DRV_NO          0       /* ドライブ番号(A) */

main()
{
        struct FREEINF  inf;    /* 残り容量情報格納領域 */

        printf("%d byte¥n", DSKFRE(DRV_NO, &inf));
        printf("free  = %d¥n", inf.free);
        printf("max = %d¥n", inf.max);
        printf("sec   = %d¥n", inf.sec);
        printf("byte  = %d¥n", inf.byte);
}
```

DUP

レベル 1

書式

```
#include <doslib.h>

int DUP (fileno) ;
int fileno; /* ファイルハンドル */
```

機能

filenoで指定されたファイルハンドルを、新しいファイルハンドルに複写します。

これは、標準入力、標準出力を切り替える（リダイレクトなど）を行う前に、切り替え前のファイルハンドルを保持しておくために使用します。

戻り値

複写された新しいファイルハンドルを返します。
負の数ならエラーを表します。

参照関数

DUP0, DUP2

DUPO

レベル 1

書　　式

```
#include <doslib.h>

int DUP0 (fileno, newno);
int fileno;  /*  複写元のファイルハンドル  */
int newno;  /*  複写先のファイルハンドル  */
```

機　　能

filenoのファイルハンドルをnewnoのファイルハンドルに強制複写します。
これは、OSがデフォルトでオープンしている0～4までのファイルハンドルの切り替えを行う（CTTY）ためのものです。
標準入出力、標準エラー出力をDUP、DUP2などで切り替えた（リダイレクトなど）あとで元にもどす場合に、CLOSEを行うことにより、元にもどります。
このときDUP0によってどのファイルハンドルにもどるかが決定されます。
newnoに指定できる番号は0～4までです。

戻り値

newnoの前の値を返します。
負の数ならエラーを表します。

参照関数

DUP, DUP2

DUP2

レベル 1

書式

```
#include <doslib.h>

int DUP2 (fileno, newno) ;
int fileno;  /*  複写元のファイルハンドル  */
int newno;   /*  複写先のファイルハンドル  */
```

機能

filenoで指定されたファイルハンドルを、newnoのファイルハンドルに強制複写します。

newnoがオープンされている場合には、クローズしてから、複写します。

これは、標準入力、標準出力を切り替えるため（リダイレクト）や、切り替えられている標準入力、標準出力を元に戻すために使用します。

戻り値

負の数ならエラーを表します。

参照関数

DUP0, DUP

EXEC2

レベル 1

書　　式

```
# include <doslib. h>

int EXEC2 (md, fil, p1, p2) ;
int md; /*  モード */
unsigned char *fil; /*  ファイル名・コマンドライン等へのポインタ */
unsigned char *p1; /*  コマンドライン等へのポインタ */
unsigned char **p2; /*  環境ポインタ等へのポインタ */
```

機　　能

プログラムのロード・実行・モジュール番号の獲得を行います。
md は以下の通りに指定します。

ビット15～8：モジュール番号
ビット7 ～ 0：モード番号

md で指定したモード番号により、動作及び引数 fil, p1, p2 の意味が違います。

1) モード番号＝0の時
p1 で指定したコマンドラインと、p2 で指定した環境ポインタを用いて、fil で指定されたパス名のファイルをロード後実行します。
p2 の環境ポインタを 0 とすると、現在の環境と同じ環境が与えられます。

2) モード番号＝1の時
p1 で指定したコマンドラインと、p2 で指定した環境ポインタを用いて、fil で指定されたパス名のファイルをロードします。
p2 の環境ポインタを 0 とすると、現在の環境と同じ環境が与えられます。

3) モード番号＝2の時
p2 で指定した環境より path を検索し、それを元に、fil で指定したコマンド行（コマンド名＋コマンドのオプションなど）をコマンドのパス名とコマンドラインに分離し、それぞれ fil と p1 の指す領域に格納します。
fil の指す領域は 90 バイト以上、また p1 の指す領域は 256 バイト以上必要です。
p2 の環境ポインタを 0 とすると、現在の環境と同じ環境が与えられます。
モード番号＝2として EXEC2 関数を実行した後に、モード番号＝0または1として EXEC2 関数を実行すれば path に設定されているディレクトリにあるコマンドを簡単に実行することができます。

4) モード番号＝3の時
p1, p2 にそれぞれロードアドレスとリミットアドレスを指定して、fil で指定したパス名のファイルをロードします。

5) モード番号 = 4の時

fil に実行アドレスを指定し、すでにロードされたプログラムを実行します。

このモードの場合 md のモジュールは0とします。

6) モード番号 = 5の時

fil で指定したオーバーレイ X ファイルより、pl で指定したファイル名のモジュールを検索し、見つかったらそのモジュール番号を返します。

このモードの場合 md のモジュール番号は0とします。

モード番号 = 0, 1, 2, 3の時、fil の最上位8ビットの意味は次の通りです。

0 : 拡張子 (.R/ .Z/ .X) にしたがい、ロードします。

1 : R タイプのファイルとしてロードします。

2 : Z タイプのファイルとしてロードします。

3 : X タイプのファイルとしてロードします。

戻り値 引数のモード番号によって異なります。

1) モード番号 = 0, 4の時

戻り値が正の数ならば、子プロセスの終了コードを表します。

負の数ならエラーを表します。

エラーの場合ロードや実行は行われません。

2) モード番号 = 1の時

戻り値が正の数ならば、ロードしたプログラムの実行アドレスを表します。

負の数ならエラーを表します。

エラーの場合ロードは行われません。

3) モード番号 = 2、3の時

負の数ならエラーを表します。

4) モード番号 = 5の時

戻り値が正の数ならば、ビット15~8にモジュール番号を返したことを表します。

負の数ならエラーを表します。

参照関数 EXECONLY, LOAD, LOADEXEC, LOADNLY, PATHCHK, BINDNO

EXECONLY

レベル 1

書　式

```
#include <doslib.h>

int EXECONLY (jmpadr);
int jmpadr; /* プログラムの実行アドレス */
```

機　能　LOAD 関数でロードしたプログラムを実行します。

戻り値　正常終了の場合には、プロセス終了コード（正の数）、異常終了したときにはエラーコード（負の数）を返します。

参照関数　LOADEXEC, LOAD, PATHCHK, LOADONLY

EXIT

レベル 1

書式

```
# include <doslib. h>

void EXIT ( ) ;
```

機能

現在のプロセスを終了させ、親プロセスに制御を返します。
また、現在オープンしているファイルをすべてクローズします。
子プロセスがオープンしたファイルについても同様です。
親プロセスに返される終了コードは0になります。

戻り値

戻り値はありません。
リターンせずに、親プロセスへ制御を移行します。

参照関数

EXIT2, KEEPPR

EXIT2

レベル 1

書　式

```
#include <doslib.h>

void EXIT2 (code) ;
int code; /*  終了コード  */
```

機　能

codeの終了コードを持ってプログラムを終了します。
この終了コードは、親プロセスに返される終了コードになります。
現在オープンされているファイルハンドルは、すべてクローズされます。
子プロセスがオープンしたファイルハンドルについても同様です。

戻り値

戻り値はありません。
リターンせずに、親プロセスへ制御を移行します。

参照関数

EXIT, KEEPPR

FATCHK

レベル 1

書式

include <doslib. h>

int FATCHK (file, buffer) ;
unsigned char ＊file ; /＊ 該当ファイルのパス名へのポインタ ＊/
unsigned short ＊buffer ; /＊ セクタ情報格納領域へのポインタ ＊/

機能

指定ファイルのドライブ番号と、セクタのつながりを調べます。

これでFATが連続していることを調べるとともに、DISKRED関数、DISKRED2関数でデータ部を直接読むこともできます（戻り値が8ならば、FATは連続しています)。

なおbufferには次に示す内容が格納されます。

unsigned short drive ; /＊ ドライブ番号 ＊/
unsigned short secno1 ; /＊ セクタの値 ＊/
unsigned short recren1 ; /＊ セクタ数 ＊/
unsigned short secno2 ; /＊ 次のセクタの値 ＊/
unsigned short recren2 ; /＊ 次のセクタ数 ＊/
⋮
(unsigned short) 0 ; /＊ 終わりを表す0 ＊/

次のセクタの値が格納されるべき所に0がセットされているならば、終わりを表します。

戻り値

bufferの使用バイト数（終わりを表す0も含む）を返します。
負の数ならエラーを表します。

参照関数

FATCHK2

プログラム例

```
#include        <doslib.h>

#define MAXCLUST        65535

static unsigned short   sinf[MAXCLUST]; /* クラスタ情報 */


main()
{
        printf("result = %d¥n", FATCHK("A:¥¥HUMAN.SYS", sinf));

        printf("ドライブ番号 = %d¥n", sinf[0]);
        printf("セクタの値   = %d¥n", sinf[1]);
        printf("セクタ数     = %d¥n", sinf[2]);
}
```

FATCHK2

レベル 1

書　式

```
# include <doslib. h>

int FATCHK2 (file, buffer, len) ;
unsigned char *file ; /* 該当ファイルのパス名へのポインタ */
unsigned short *buffer ; /* セクタ情報格納領域へのポインタ */
int len ; /* バッファの最大長 */
```

機　能

指定ファイルのドライブ番号と、セクタの連続性を調べます。
FATCHK 関数に対して、FATCHK2 関数は大容量ドライブ・特殊ドライブに対応しています（戻り値が 14 ならば、FAT は連続しています）。
buffer には、次に示す内容が格納されます。

```
unsigned short drive ;    /* ドライブ番号 */
unsigned long secnol ;    /* 先頭のセクタ番号 */
unsigned long seclenl ;   /* セクタ数 */
unsigned long secnol ;    /* 次のセクタ番号 */
unsigned long seclenl ;   /* セクタ数 */
              :
(unsigned long) ;         /* 終わりを表す 0 */
```

次のセクタの値が格納されるべき所に 0 がセットされているならば終わりを表します。

戻り値

buffer の使用バイト数（終わりを表す 0 も含む）を返します。
負の数ならエラーを表します。

参照関数

FATCHK

プログラム例

```
#include        <stdio.h>
#include        <doslib.h>

#define SIZE    1024

typedef unsigned long   N_t;


main()
{
        static char     buf[SIZE];
        char    *inf;
        N_t     s;
        N_t     n;
        N_t     i;
        int     r;


        r = FATCHK2("/config.sys", buf, SIZE);

        inf = buf;
        printf("Drive No. = %d¥n", *((unsigned short *)inf));
        for (inf += 2; (s = *((N_t *)inf)) != 0L; inf += 4) {
                inf += 4;
                n = *((N_t *)inf);
                for (i = 0; i < n; i++)
                        printf("%ld ", s + i);
        }
        printf("¥n");
        if (r == 14)
                printf("クラスタは連続しています。¥n");
}
```

FFLUSH

レベル 1

書　式

```
#include <doslib.h>

void FFLUSH ( ) ;
```

機　能

ディスクのリセットを行います。
ディスクバッファの内容はすべてフラッシュしますが、ファイルのクローズはしません。

戻り値

戻り値はありません。

参照関数

flushall

FGETC

レベル 1

書　式　#include <doslib.h>

```
int FGETC (fileno) ;
int fileno ;  /*  ファイルハンドルの番号  */
```

機　能　指定されたファイルハンドルから入力があるまで待ち、1バイト入力します。

戻り値　入力コードを返します。

FGETS

レベル1

書式

```
#include <doslib. h>

int FGETS (inpptr, fileno) ;
struct INPPTR  { /*  入力文字列格納領域  */
  unsigned char max ;  /*  1行入力時の最大文字数  */
  unsigned char length ;  /*  1行入力が行われた文字数  */
  unsigned char buffer [256] ;  /*  キー入力データ  */
} *inpptr ;
int fileno ;
```

機能

filenoで指定されたファイルハンドルからCRコードが入力されるまで、文字列を入力します。
入力最大文字数を超えた場合は、入力最大文字数までを入力します。
lengthには実際に1行入力が行われた文字数が、bufferには入力データが格納されます。

戻り値

CRコードを除いた入力文字数を返します。

戻り値

GETS, GETSS

FILEDATE

レベル 1

書　式

```
#include <doslib.h>

int FILEDATE (fileno, datetime) ;
int fileno;  /* 対象とするファイルのファイルハンドル */
int datetime; /* 日付と時刻 */
```

機　能

filenoで指定したファイルの日付/時刻の読み出しと設定を行います。
datetimeに0を指定したときは読み出しを行い、これ以外の値を指定したときには設定を行います。
datetime、および戻り値の形式は次の通りです。

datetime = YYYYYYYM MMMDDDDD hhhhhmmm mmmsssss

ビット 0〜4	：	(sssss)	0〜29秒（実際の値はこれに2を乗ずる）
ビット 5〜10	：	(mmmmmm)	0〜59分
ビット 11〜15	：	(hhhhh)	0〜23時
ビット 16〜20	：	(DDDDD)	1〜31日
ビット 21〜24	：	(MMMM)	1〜12月
ビット 25〜31	：	(YYYYYYY)	0〜99年（1980年から相対年数）

日付/時刻の設定を行う場合には、filenoで指定されたファイルハンドルは、書き込みが可能なモードでオープンされていなければなりません。
また、日付/時刻の設定を行ったあとで、ファイルハンドルに対して書き込みを行ってしまうと、設定された日付/時刻は無効になります。
必ずクローズを行う直前で、日付/時刻の設定を行ってください。

戻り値

datetimeが0の場合のときのみ、読み出したファイルの日付/時刻を返します。
上位16ビットが0xFFFFならばエラーを表します。

プログラム例

```
#include        <stdio.h>
#include        <fcntl.h>
#include        <stat.h>
#include        <io.h>
#include        <doslib.h>

main()
{
        int     fno;                /* ファイルハンドル */
        int     datetime;           /* 日付と時刻 */
        int     r;                  /* 戻り値 */


        fno = open("A:¥¥CONFIG.SYS", O_RDONLY);

        datetime = 0;               /* 日付と時刻の読出し */
        r = FILEDATE(fno, datetime);

        printf("r      = %032b¥n", r);
        printf("second = %d¥n", ((r >> 0) & 0x1f) * 2);
        printf("minute = %d¥n", ((r >> 5) & 0x3f));
        printf("hour   = %d¥n", ((r >> 11) & 0x1f));
        printf("day    = %d¥n", ((r >> 16) & 0x1f));
        printf("month  = %d¥n", ((r >> 21) & 0x0f));
        printf("year   = %d¥n", ((r >> 25) & 0x7f) + 1980);
}
```

FILES

レベル 1

書　式

```
#include <doslib.h>

int FILES (buffer, file, atr) ;
struct FILBUF *buffer ; /* ファイル情報格納領域へのポインタ */
unsigned char *file ; /* ファイル名格納領域へのポインタ */
int atr ; /* ファイル属性 */
struct FILBUF {
  unsigned char OS[21] ; /* Human 68k が内部で使用します */
  unsigned char atr ; /* ファイル属性 */
  unsigned short time ; /* ファイルの時刻 */
  unsigned short date ; /* ファイルの日付 */
  unsigned int filelen ; /* ファイルの長さ */
  unsigned char name[23] ; /* ファイル名 */
} ;
```

機　能

file で指定され、atr の属性をもつファイルをサーチして、その情報を buffer へ格納します。

ファイル名にはワイルドカード文字が使えます。

ワイルドカードを指定したときに、最初に見つかったファイルが目的のファイルでない場合、NFILES で次のファイルをサーチすることができます。

atr に指定する属性は次の通りです。

0x01　：読み込み専用
0x02　：隠しファイル
0x04　：システムファイル
0x08　：ボリューム ID
0x10　：ディレクトリ
0x20　：通常ファイル

この atr で示される属性と、ファイルの属性との論理積（AND）を取った値が0でなければサーチの対象となります。

したがって、すべての属性をサーチの対象にする場合は 0x3F を指定します。

また、読み込み専用の通常ファイルをサーチ対象としようとして 0x21 を指定しても、すべての通常ファイルとすべての読み込み専用ファイルが対象となってしまうことに注意してください。

戻り値　正の数（0も）で正常終了、負の数はエラーを表します。
通常は0が返りますが、特殊デバイスドライバが対象の時0以外の正の数の場合があります。

参照関数　NFILES

プログラム例

```
#include        <stdio.h>
#include        <doslib.h>

main()
{
        struct FILBUF   inf;     /* ファイル情報 */
        int     atr;             /* ファイル属性 */

        atr = 0x20;              /* 通常ファイル */

        if (FILES(&inf, "*.*", atr) >= 0) {  /* 1つめがあれば */
                printf("%s¥n", inf.name);        /* それを表示 */

                while (NFILES(&inf) >= 0)/* 2つめ以降がある間 */
                        printf("%s¥n", inf.name); /* 表示する */
        }
}
```

FILL

レベル 0

書式

```
#include <iocslib.h>

int FILL (fillptr) ;
struct FILLPTR {
  short x1 ; /* 始点の x 座標 */
  short y1 ; /* 始点の y 座標 */
  short x2 ; /* 終点の x 座標 */
  short y2 ; /* 終点の y 座標 */
  unsigned short color ; /* パレットコード */
} *fillptr ;
```

機能

グラフィック画面にぬりつぶした四角形を描きます。

戻り値

終了コードを返します。

0 ： 正常終了
−1 ： グラフィックは使用不可

FNCKEYGT

レベル 1

書　　式

```
# include <doslib. h>

void FNCKEYGT (fno, buffer) ;
int fno ;  /*  定義可能キー番号  */
unsigned char *buffer ;  /*  読み出し用データバッファへのポインタ  */
```

機　　能

fnoで指定した再定義可能キーからキーの内容を読み出します。
このファンクションコールは、CONデバイスによりサポートされます。
fnoで指定するキー、およびデータバッファの必要サイズは次の通りです。

● fno

0	：	すべてのキー
1〜10	：	F1〜F10
11〜20	：	SHIFT+F1〜F10
21	：	ROLL UP
22	：	ROLL DOWN
23	：	INS
24	：	DEL
25	：	UP
26	：	LEFT
27	：	RIGHT
28	：	DOWN
29	：	CLR
30	：	HELP
31	：	HOME
32	：	UNDO

● buffer

fno	：0のとき	20×32+12×6 (712) バイト
fno	：1〜20のとき	32バイト (31バイト+¥0)
fno	：21〜32のとき	6バイト (5バイト+¥0)

戻り値

戻り値はありません。

参照関数

FNCKEYST

プログラム例

```
#include        <stdio.h>
#include        <doslib.h>

#define F10     10      /* F10 */
#define F20     20      /* shift+F10 */

main()
{
        char    buf[32];        /* データバッファ */

        FNCKEYGT(F10, buf);     /* F10読み出し */
        printf("F10: %s¥n", buf + 1);

        FNCKEYGT(F20, buf);     /* shift+F10読み出し */
        printf("F20: %s¥n", buf);
}
```

FNCKEYST

レベル 1

書　式

```
#include <doslib.h>

void FNCKEYST (fno, buffer) ;
int fno ;  /*  再定義可能キー番号  */
unsigned char *buffer ;  /*  設定用データバッファへのポインタ  */
```

機　能

fnoで指定した再定義可能キーの設定を行います（同時に、ファンクションキーの表示も行います。ただし、32行モードのときには表示しません）。
このファンクションコールは、CONデバイスによりサポートされます。
fnoで指定できるキー、およびデータバッファの必要なバイト数は次の通りです。

● fno

0	：	すべてのキー
1～10	：	F1～F10
11～20	：	SHIFT＋F1～F10
21	：	ROLL UP
22	：	ROLL DOWN
23	：	INS
24	：	DEL
25	：	UP
26	：	LEFT
27	：	RIGHT
28	：	DOWN
29	：	CLR
30	：	HELP
31	：	HOME
32	：	UNDO

● buffer

fno	： 0のとき	20×32＋12×6（712）バイト
fno	： 1～20のとき	32バイト（31バイト＋¥0）
fno	： 21～32のとき	6バイト（5バイト＋¥0）

戻り値

戻り値はありません。

参照関数

FNCKEYGT

プログラム例

```
#include        <stdio.h>
#include        <doslib.h>

#define F9      9

main()
{
        char    buf[32];                /* データバッファ */

        FNCKEYGT(F9, buf);              /* 退避 */

        FNCKEYST(F9, "FNCKEY");         /* 設定 */

        puts("キーを押すともとに戻ります。");
        while (K_KEYSNS() == 0)
                ;

        FNCKEYST(F9, buf);              /* 元に戻す */
}
```

FNTGET

レベル0

書　式

```
#include <iocslib.h>

int FNTGET (TYPE, CODE, BUF);
int TYPE;  /*  文字の大きさ  */
int CODE;  /*  漢字コード  */
struct FNTBUF *BUF /*  データバッファアドレス  */
```

機　能

CODEで指定された漢字パターンを、BUFで指定されたアドレスへ読み込みます。

BUFにはスーパーバイザ領域のアドレスも指定できますが、誤ったアドレスを指定しないように注意してください。

TYPEには8または12を指定します。

0を指定した場合は8と同じです。

CODEにはシフトJIS漢字コードを指定しますが、JIS漢字コード（$2121～$7E7E）を指定することもできます。

BUFには、データバッファの先頭アドレスを指定します。

この領域は、TYPEに8または0を指定した時には36バイトの、TYPEに12を指定した時には76バイトの領域が必要です。

構造体FNTBUFのデータサイズを越える場合には、メモリ確保関数mallocなどで確保した領域を使用してください。

BUFには、漢字パターンのX方向のドット数（short）、Y方向のドット数（short）、およびパターンデータが格納されます。

戻り値

戻り値はありません。

プログラム例

```
#include        <stdio.h>
#include        <iocslib.h>

static struct FNTBUF    fnt;    /* データバッファ */

main()
{
        int     type;           /* 文字の大きさ */
        int     code;           /* 漢字コード */

        type = 8;
        code = 0x989f;

        FNTGET(type, code, &fnt);

        printf("x: %d dots¥n", fnt.xl);
        printf("y: %d dots¥n", fnt.yl);

        code = 0xebe9;
        DEFCHR(type, code, fnt.buffer);
        printf("%c%c¥n", 0xeb, 0xe9);
}
```

FPUTC

レベル 1

書　式

```
#include <doslib.h>

void FPUTC (code, fileno) ;
int code ;  /*  出力する文字コード  */
int fileno ;  /*  出力するファイルハンドル  */
```

機　能

指定した文字コードをファイルハンドルに出力します。
出力デバイスがキャラクタデバイスの場合には、ブレークチェックを行います（^C：^S：^P：^N：）。

戻り値

戻り値はありません。

FPUTS

レベル 1

書　式

```
#include <doslib.h>

void FPUTS (msgptr, fileno) ;
unsigned char *msgptr;  /*  出力する文字列へのポインタ  */
int fileno;  /*  出力するファイルハンドル  */
```

機　能

ヌル文字で終了する文字列を、ファイルハンドルに出力します。
出力するデバイスがキャラクタデバイスの場合には、ブレークチェックを行います（^C：^S：^P：^N：）。

戻り値

戻り値はありません。

G_CLR_ON

レベル 0

書　式

```
#include <iocslib.h>

void G_CLR_ON ( ) ;
```

機　能

グラフィック画面をクリアして、表示モードにします。
グラフィックパレットは初期状態に戻ります。

戻り値

戻り値はありません。

GETASSIGN　レベル1

書　式

```
# include <doslib.h>

int GETASSIGN (buff1, buff2) ;
unsigned char *buff1; /* ドライブ名 */
unsigned char *buff2; /* ディレクトリ名を格納するポインタ */
```

機　能

仮想ドライブ、仮想ディレクトリの割り当て状況を取得します。
戻り値によって、以下のような割り当て状況がbuff2へセットされます。

- 戻り値が0x40のとき（割り当てが存在しない）
  buff1で指定されたドライブのカレントディレクトリがbuff2へセットされます。
- 戻り値が0x50のとき（仮想ドライブの割り当て）
  buff1をアクセスしたとき、実際にアクセスされるディレクトリがbuff2へセットされます。
- 戻り値が0x60のとき（仮想ディレクトリの割り当て）
  buff1を実際にアクセスしようとしたとき、アクセスしなければならないディレクトリがbuff2にセットされます。

buff1に、MAKEASSIGN関数での1番目のパラメータと同じ内容のパラメータを指定すると、2番目のパラメータで指定していた内容がbuff2にセットされます。

戻り値

以下に示す割り当てモードを返します。

0x40 : 割り当てが存在しない
0x50 : 仮想ドライブの割り当て
0x60 : 仮想ディレクトリの割り当て

負の数ならエラーを表します。

参照関数

MAKEASSIGN, RASSIGN

GETC

レベル 1

書　式　# include <doslib. h>

```
int GETC ( ) ;
```

機　能　標準入力から1文字読み込みます。
このときブレークチェックを行います（^C：^P：^N：)。

戻り値　読み込んだ文字コードを返します。

参照関数　INKEY, GETCHAR, getc

GETCHAR

レベル 1

書　式	#include <doslib.h> int GETCHAR ( );
機　能	標準入力から1文字読み込み、読み込んだ文字を標準出力にエコーバックします。このときブレークチェックも行います（^C：^P：^N：)。
戻り値	読み込んだ文字コードを返します。
参照関数	PUTCHAR, GETC, INPOUT, INKEY, getche

GETDATE

レベル 1

書　式　# include <doslib. h>

int GETDATE ( ) ;

機　能　現在の日付を調べます。

戻り値　現在の日付を返します。
形式は次の通りです。

00000000 00000www yyyyyyym mmmddddd

ビット 0～4 (ddddd)　：日 (1～31)
ビット 5～8 (mmmm)　：月 (1～12)
ビット 9～15 (yyyyyyy)　：年 (0～99, 1980年からの相対年数)
ビット16～18 (www)　：曜日 (0 ＝ 日・・・6 ＝ 土)

参照関数　GETTIM2, SETTIM2, SETDATE, GETTIME, SETTIME

プログラム例

```
#include        <stdio.h>
#include        <doslib.h>

main()
{
        int     date;
        int     yy;
        int     mm;
        int     dd;

        date = GETDATE();

        yy = ((date >> 9) & 0x7f) + 1980;
        mm = ((date >> 5) & 0x0f);
        dd = ((date >> 0) & 0x1f);
        printf("%d年%d月%d日¥n", yy, mm, dd);
}
```

GETDPB

レベル 1

書式

```
#include <doslib.h>

int GETDPB (drive, dpbptr) ;
int drive ; /* ドライブ番号 */
struct DPBPTR *dpbptr ; /* ドライブパラメータブロック格納領域 */
```

機能

指定したドライブ番号のドライブパラメータブロックを指定した格納領域へ格納します。

driveに指定できるデバイス番号、ドライブパラメータブロックの形式は、それぞれ次の通りです。

●ドライブ番号

0：カレントドライブ
1：ドライブA
2：ドライブB
⋮

●ドライブパラメータブロック

```
struct DPBPTR {
unsigned char drive;       /* ドライブ番号 (0 = A, 1 = B) */
unsigned char unit;        /* デバイスドライバで使うユニット番号 */
unsigned short byte;       /* 1セクタ数あたりのバイト数 */
unsigned char sec;         /* 1クラスタあたりのセクタ数−1 */
unsigned char shift;       /* クラスタ、セクタ間のシフト数 */
unsigned short fatsec;     /* FATの先頭セクタ番号 */
unsigned char fatcount;    /* FAT領域の個数 */
unsigned char fatlen;      /* FATの占めるセクタ数 (コピー分は別) */
unsigned short dircount;   /* ルートディレクトリの個数 */
unsigned short datasec;    /* データ部先頭のセクタ番号 */
unsigned short maxfat;     /* 総クラスタ数+1 */
unsigned short dirsec;     /* ルートディレクトリの先頭セクタ番号 */
int driver;                /* デバイスドライバへのポインタ */
unsigned char id;          /* メディアバイト */
unsigned char flg;         /* dpb使用フラグ (−1のときアクセスなし) */
struct DPBPTR *next;       /* 次の dpb へのポインタ */
```

```
unsigned short dirfat;       /*  カレントディレクトリのクラスタ番号  */
                             /*  (0のときはルートディレクトリを表す)  */
unsigned char dirbuf[64];    /*  カレントディレクトリの文字バッファ  */
};
```

戻り値　負の数ならエラーを表します。

プログラム例

```
#include        <stdio.h>
#include        <doslib.h>

main()
{
        struct DPBPTR   dpb;        /* ドライブパラメータブロック */
        int     drive;              /* ドライブ番号 */

        drive = 0;

        GETDPB(drive, &dpb);

        printf("drive: %d¥n", dpb.drive);
        printf("unit : %d¥n", dpb.unit);
        printf("byte : %d¥n", dpb.byte);
        printf("sec  : %d¥n", dpb.sec);
}
```

GETENV　レベル 1

書　式　# include <doslib. h>

int GETENV (name, env, buffer) ;
unsigned char *name; /* 環境変数へのポインタ */
unsigned char *env; /* 環境のポインタ */
unsigned char *buffer; /* 環境変数格納領域へのポインタ */

機　能　envで指定された環境から、nameで指定した環境変数の内容をとり出します。
envに0を指定すると、親プロセスの環境からとり出します。
環境変数の内容はbufferに格納されますが、この領域は256バイト必要です。

戻り値　負の数ならエラーを表します。

参照関数　SETENV

プログラム例

```
#include        <stdio.h>
#include        <doslib.h>

main()
{
        char    *name;          /* 環境変数 */
        char    *env = NULL;    /* 環境 */
        char    buf[256];       /* 環境変数格納領域 */

        name = "path";

        GETENV(name, env, buf);

        printf("%s¥n", buf);
}
```

GETGRM

レベル0

書　式

```
#include <iocslib.h>

int GETGRM (getptr) ;
struct GETPTR {
  short x1 ;  /*  始点のX座標  */
  short y1 ;  /*  始点のY座標  */
  short x2 ;  /*  終点のX座標  */
  short y2 ;  /*  終点のY座標  */
  unsigned char *buf_start ;/*  バッファ先頭アドレス  */
  unsigned char *buf_end ;  /*  バッファ終了アドレス  */
} *getptr ;
```

機　能

グラフィック画面からの読み込みを行います。
バッファには、データがパックされた型で格納されます。

戻り値

終了コードを返します。

　0 ：正常終了
−1 ：グラフィックは使用不可

GETPDB

レベル 1

書　式

```
# include <doslib.h>

struct PDBADR  *GETPDB ( ) ;

struct PDBADR {
  unsigned int env ; /*  プロセスに与えられた環境のアドレス */
  unsigned int exit ; /*  プロセスが終了したさいの戻りアドレス */
  unsigned int ctrlc ; /*  プロセスがCTRL+Cにより中断された際の戻り
                        アドレス */
  unsigned int errexit ; /*  プロセスがエラーにより中止された際の戻りア
                          ドレス */
  unsigned int comline ; /*  プロセスに与えられたコマンドラインのアドレ
                          ス */
  unsigned char handle[12] ; /*  プロセスのファイルハンドラの使用状況
                              */
  unsigned int bss ; /*  プロセスのbssの先頭アドレス */
  unsigned int heep ; /*  プロセスのヒープの先頭アドレス */
  unsigned int stack ; /*  プロセスの初期スタックアドレス */
  unsigned int usp ; /*  親プロセスのUSPレジスタの値 */
  unsigned int ssp ; /*  親プロセスのSSPレジスタの値 */
  unsigned short sr ; /*  親プロセスのSRレジスタの値 */
  unsigned short abort_sr ; /*  アボート（中止）のときのSRレジスタの値
                             */
  unsigned int abort_ssp ; /*  アボート（中止）のときのSSPレジスタの
                            値 */
  unsigned int trap10 ; /*  TRAP10 */
  unsigned int trap11 ; /*  TRAP11 */
  unsigned int trap12 ; /*  TRAP12 */
  unsigned int trap13 ; /*  TRAP13 */
  unsigned int trap14 ; /*  TRAP14 */
  unsigned int osflag ; /*  プロセスのフラグ、0で親あり、−1でOSから
                         起動 */
  unsigned char reserve[28] ; /*  未使用 */
  unsigned char exe_path[68] ; /*  execされたファイルのパス名 */
  unsigned char exe_name[24] ; /*  execされたファイル名 */
} ;
```

機　　能　現在のプロセスを表す pdbadr を求めます。
pdbadr はプログラムの先頭アドレス−0xF0 です。
なお、求めた pdbadr は SETPDB 関数で使用します。

戻 り 値　現在のプロセスを表す pdbadr の値を返します。

参照関数　SETPDB

GET_PR

レベル 1

書式

```
# include <doslib. h>

int GET_PR (id, buff) ;
int id;                            /* スレッドのID */
struct PRCPTR {                    /* スレッド管理情報格納領域 */
  struct PRCPTR * next_ptr;        /* 次の管理領域へのポインタ */
  unsigned char wait_flg;          /* normal = $00 / wait = $ff */
  unsigned char counter;           /* 割り込みの度に減算される */
  unsigned char max_counter;       /* counter のプリセット値 */
  unsigned char doscmd;
  unsigned int psp_id;             /* プロセス ID */
  unsigned int usp_reg;            /* usp 格納領域 */
  unsibned int d_reg [8] ;         /* データレジスタ格納領域 */
  unsigned int a_reg [7] ;         /* アドレスジスタ格納領域 */
  unsigned short sr_reg;           /* sr 格納領域 */
  unsigned int pc_reg;             /* pc 格納領域 */
  unsigned int ssp_reg;            /* ssp 格納領域 */
  unsigned short indosf;
  unsigned int indosp;
  struct PRCCTRL * buf_ptr;        /* タスク間通信バッファへのポインタ */
  unsigned char name [16] ;        /* スレッドの名前 */
  long wait_time;                  /* 待ち時間の残り (ミリ秒) */
} *buff;
```

機能

指定されたIDのバックグラウンドタスクの管理情報をbuffにコピーします。
idに−1を指定し、buff−>nameにバックグラウンドタスクの名前を指定してGET_PR関数を呼び出すと、戻り値としてバックグラウンドタスクのidが返り、管理情報がbuffにコピーされます。
idに−2を指定すると、自分自身のIDが戻り値として返り、管理情報がbuffにコピーされます。

戻り値	求めるバックグラウンドタスクのidを返します。 戻り値が負の数の場合はエラーコードを示します。 指定したidがおかしい場合は、0xFFFF00?? (??は不定の16進数を表します）を返します。 0x??がidの最大値です。 指定した名前のバックグラウンドタスクが存在しない場合は－1を返します。
参照関数	OPEN_PR, KILL_PR, SUSPEND_PR, SLEEP_PR, SEND_PR, TIME_PR, CHANGE_PR

GETS

レベル 1

書式

```
#include <doslib.h>

int GETS (inpptr);
  struct INPPTR { /* 入力文字列格納領域 */
  unsigned char max; /* 入力時の最大文字数 */
  unsigned char length; /* 入力が行われた文字数 */
  unsigned char buffer [256]; /* 入力データ */
} *inpptr;
```

機能

CR コードが入力データに現れるまで、文字列を入力します。
このときブレークチェックも行います（^C：^P：^N：）。
入力文字数が最大文字数 inpmax を超えた場合は、警告音を出しますが終了はしません。
length には実際に入力が行われた文字数が、buffer には入力データが格納されます。

戻り値

CR コードを除く入力文字数を返します。

参照関数

PRINT, GETSS, FGETS

GETSS

レベル 1

書　　式

```
#include <doslib.h>

int GETSS (inpptr);
  struct INPPTR {   /*  入力文字列格納領域  */
  unsigned char max;  /*  入力時の最大文字数  */
  unsigned char length;  /*  入力が行われた文字数  */
  unsigned char buffer [256];  /*  入力データ  */
} *inpptr;
```

機　　能

CR コードが入力データに現れるまで、文字列を入力します。
ただし、ブレークチェックは行いません。
また、VOID/NEWLINE で改行しません。
入力文字数が入力最大文字数 inpmax を超えた場合は警告音を出しますが、終了はしません。
length には実際に入力が行われた文字数が、buffer には入力データが格納されます。

戻 り 値

CR コードを除いた入力文字数を返します。

参照関数

GETS, FGETS

GETTIM2

レベル 1

書　式　# include <doslib. h>

int GETTIM2 ( ) ;

機　能　現在の時刻を調べます。

戻り値　現在の時刻を返します。
形式は次の通りです。

00000000 000hhhhh 00mmmmmm 00ssssss

ビット 0～5 (ssssss) ： 秒 (0～59)
ビット 8～13 (mmmmmm) ： 分 (0～59)
ビット 16～20 (hhhhh) ： 時 (0～23)

参照関数　SETTIM2, GETDATE, SETDATE, GETTIME, SETTIME

プログラム例

```
#include        <stdio.h>
#include        <doslib.h>

main()
{
        int     time;   /* 現在の時刻 */
        int     hh;
        int     mm;
        int     ss;

        time = GETTIM2();

        hh = (time >> 16) & 0x1f;
        mm = (time >> 8) & 0x3f;
        ss = (time >> 0) & 0x3f;
        printf("%d時%d分%d秒¥n", hh, mm, ss);
}
```

GETTIME

レベル 1

書式 # include <doslib. h>

int GETTIME ( ) ;

機能 現在の時刻を調べます。

戻り値 現在の時刻を返します。
形式は次の通りです。

hhhhhmmm mmmsssss

ビット 0〜4 (sssss) ： 秒 (0〜29, 実際の値は 2 を乗ずる)
ビット 5〜10 (mmmmmm) ： 分 (0〜59)
ビット 11〜15 (hhhhh) ： 時 (0〜23)

参照関数 GETTIM2, SETTIM2, GETDATE, SETDATE, SETTIME

プログラム例

```
#include        <stdio.h>
#include        <doslib.h>

main()
{
        int     time;   /* 現在の時刻 */
        int     hh;
        int     mm;
        int     ss;

        time = GETTIME();

        hh = ((time >> 11) & 0x1f);
        mm = ((time >> 5) & 0x3f);
        ss = ((time >> 0) & 0x1f) * 2;
        printf("%d時%d分%d秒¥n", hh, mm, ss);
}
```

GPALET

レベル 0

書　式

include <iocslib. h>

int GPALET (PALET_NO, COLOR_CODE) ;
int PALET_NO ; /＊ パレット番号 ＊/
int COLOR_CODE ; /＊ カラーコード ＊/

機　能

グラフィックのパレットを設定します。
PALET_NO にはパレット番号 (0～15) / (0～255) / (0～65535) を指定します。
COLOR CODE にはカラーコード (0～65535) を指定します。
−1 を指定するとパレット番号の読み出しのみ行います。

戻り値

COLOR_CODE に−1 を指定したときのみ、現在のカラーコードを返します。

プログラム例

```
#include        <stdio.h>
#include        <iocslib.h>

#define PALET           12
#define COLOR           216

main()
{
        int     palet_no;       /* パレット番号 */
        int     color_code;     /* カラーコード */

        palet_no = PALET;
        color_code = COLOR;

        G_CLR_ON();

        printf("変更前: %d¥n", GPALET(palet_no, -1));
        GPALET(palet_no, color_code);
        printf("変更後: %d¥n", GPALET(palet_no, -1));

        G_CLR_ON();
}
```

HANJOB

レベル0

書　式

include <iocslib. h>

int HANJOB (BUFEND) ;
unsigned char ＊BUFEND;　/＊　ヌル文字のアドレス　＊/

機　能

半濁点処理を行います。
BUFENDには、全角文字列の終端にあるヌル文字のアドレスを指定します。
最後の全角文字に半濁点処理をできない時は、文字列に（゜）を追加しますので、ヌル文字の後に3バイト分の領域が必要です。
また、（゜）を追加した場合は（゜）の後にヌル文字を付けますので、文字列の途中を指定すると（゜）の後の文字列が切れてしまいます。
したがって、文字列の途中を指定しないでください。

戻り値

次の内容を返します。

0：最後の全角文字に半濁点処理をした
2：（゜）を追加した

参照関数

DAKJOB

プログラム例

```
#include        <stdio.h>
#include        <iocslib.h>

static char     str[13] = "はひふへほ"; /* 全角文字列バッファ */

main()
{
        char    *endp;          /* バッファ最終アドレス */

        printf("%s¥n", str);
        endp = str + strlen(str);

        HANJOB(endp);

        printf("%s¥n", str);
}
```

HENDSPIC

レベル 1

書　式

```
#include <doslib.h>

void HENDSPIC (position);
int position; /* ウィンドウ上の表示位置 */
```

機　能

position 以降の内容をもとに戻します。
このファンクションコールは、漢字変換フロントプロセッサのためのものであり、ふつうのアプリケーションプログラムで使用してはいけません。

戻り値

戻り値はありません。

参照関数

HENDSPIO, HENDSPIP, HENDSPIR

HENDSPIO

レベル 1

書　式

```
#include <doslib.h>

int HENDSPIO ( ) ;
```

機　能

入力ウィンドウをオープンします。
このファンクションコールは、漢字変換フロントプロセッサのためのものであり、ふつうのアプリケーションプログラムで使用してはいけません。

戻り値

ウィンドウの最大表示文字数を返します。

参照関数

HENDSPIC, HENDSPIP, HENDSPIR

HENDSPIP

レベル 1

書式 #include <doslib.h>

```
int HENDSPIP (position, msgptr) ;
int position ;  /*  ウィンドウ上の表示位置  */
unsigned char *msgptr ;  /*  メッセージ格納領域へのポインタ  */
```

機能 メッセージを、入力ウィンドウにノーマルで表示します。
このファンクションコールは、漢字変換フロントプロセッサのためのものであり、ふつうのアプリケーションプログラムで使用してはいけません。

戻り値 ウィンドウでの次の表示位置を返します。

参照関数 HENDSPIC, HENDSPIO, HENDSPIR

HENDSPIR

レベル 1

書　　式

```
#include <doslib.h>

int HENDSPIR (position, msgptr) ;
int position; /*　ウィンドウ上の表示位置　*/
unsigned char *msgptr; /*　メッセージ格納領域へのポインタ　*/
```

機　　能

メッセージを入力ウィンドウにリバースで表示します。
このファンクションコールは、漢字変換フロントプロセッサのためのものであり、ふつうのアプリケーションプログラムで使用してはいけません。

戻 り 値

ウィンドウでの次の表示位置を返します。

参照関数

HENDSPIC, HENDSPIO, HENDSPIP

HENDSPMC

レベル 1

書　　式　# include <doslib. h>

```
void HENDSPMC ( ) ;
```

機　　能　モード表示ウィンドウをクローズします。
このファンクションコールは、漢字変換フロントプロセッサのためのものであり、ふつうのアプリケーションプログラムで使用してはいけません。

戻 り 値　戻り値はありません。

参照関数　HENDSPMO, HENDSPMP, HENDSPMR

HENDSPMO

レベル 1

書　式

```
#include <doslib.h>

int HENDSPMO ( ) ;
```

機　能

モード表示ウィンドウをオープンします。
このファンクションコールは、漢字変換フロントプロセッサのためのものであり、ふつうのアプリケーションプログラムで使用してはいけません。

戻り値

ウィンドウでの最大表示文字数を返します。

参照関数

HENDSPMC, HENDSPMP, HENDSPMR

HENDSPMP

レベル 1

書　式

```
#include <doslib.h>

int HENDSPMP (position, msgptr) ;
int position ;  /* ウィンドウ上の表示位置 */
unsigned char *msgptr ;  /* メッセージ格納領域へのポインタ */
```

機　能

メッセージを、モード表示ウィンドウにノーマルで表示します。
このファンクションコールは、漢字変換フロントプロセッサのためのものであり、ふつうのアプリケーションプログラムで使用してはいけません。

戻り値

ウィンドウ上の次の表示位置を返します。

参照関数

HENDSPMC, HENDSPMO, HENDSPMR

HENDSPMR

レベル 1

書　式

```
#include <doslib.h>

int HENDSPMR (position, msgptr);
int position; /* ウィンドウ上の表示位置 */
unsigned char *msgptr; /* メッセージ格納領域へのポインタ */
```

機　能

メッセージを、モード表示ウィンドウにリバースで表示します。
このファンクションコールは、漢字変換フロントプロセッサのためのものであり、ふつうのアプリケーションプログラムで使用してはいけません。

戻り値

ウィンドウ上の次の表示位置を返します。

参照関数

HENDSPMC, HENDSPMO, HENDSPMP

HENDSPSC

レベル 1

書　　式　# include <doslib. h>

```
void HENDSPSC ( ) ;
```

機　　能　候補ウィンドウをクローズします。
このファンクションコールは、漢字変換フロントプロセッサのためのものであり、ふつうのアプリケーションプログラムで使用してはいけません。

戻 り 値　戻り値はありません。

参照関数　HENDSPSO, HENDSPSP, HENDSPSR

HENDSPSO

レベル 1

書　　式　# include <doslib. h>

```
int HENDSPSO ( ) ;
```

機　　能　候補ウィンドウをオープンします。
このファンクションコールは、漢字変換フロントプロセッサのためのものであり、ふつうのアプリケーションプログラムで使用してはいけません。

戻 り 値　ウィンドウの最大表示文字数を返します。

参照関数　HENDSPSC, HENDSPSP, HENDSPSR

HENDSPSP

レベル 1

書式

```
#include <doslib.h>

int HENDSPSP (position, msgptr) ;
int position; /* ウィンドウ上の表示位置 */
unsigned char *msgptr; /* メッセージ格納領域へのポインタ */
```

機能

メッセージを、候補ウィンドウにノーマルで表示します。
このファンクションコールは、漢字変換フロントプロセッサのためのものであり、ふつうのアプリケーションプログラムで使用してはいけません。

戻り値

ウィンドウ上の次の表示位置を返します。

参照関数

HENDSPSC, HENDSPSO, HENDSPSR

HENDSPSR

レベル 1

書　式

```
#include <doslib.h>

int HENDSPSR (position, msgptr) ;
int position ;  /*  ウィンドウ上の表示位置  */
unsigned char *msgptr ;  /*  メッセージ格納領域へのポインタ  */
```

機　能

メッセージを、候補ウィンドウにリバースで表示します。
このファンクションコールは、漢字変換フロントプロセッサのためのものであり、ふつうのアプリケーションプログラムで使用してはいけません。

戻り値

ウィンドウ上の次の表示位置を返します。

参照関数

HENDSPSC, HENDSPSO, HENDSPSP

HOME

レベル 0

書式 #include <iocslib.h>

```
int HOME (PAGE, X, Y) ;
int PAGE;  /*  ページ指定  */
int X;  /*  X座標  */
int Y;  /*  Y座標  */
```

機能 グラフィック画面の表示位置を設定します。
設定するページはPAGEのビット0～ビット3で指定します（たとえば、ページ0を指定するときはビット0をONにします）。
ビット0～ビット3がすべて0ならば、現在のモードで有効なページがすべて設定されます。
XにはX座標、YにはY座標を指定します。
ただし指定できるのはいずれも0～1023までです。

戻り値 終了コードを返します。

0：正常終了
−1：グラフィックは使用不可
−2：引数が指定通りになっていない
−3：現在のモードでは設定不可

HSVTORGB

レベル 0

書　式

```
#include <iocslib.h>

int HSVTORGB (H, S, V) ;
int H; /* カラー */
int S; /* 白レベル */
int V; /* 黒レベル */
```

機　能

HSV 方式から RGB 方式に変換します。
H で指定できるカラーは次の通りです。
なお、それぞれの間は 32 段階に分かれていますので色合いが指定できます。

0〜31	：	赤〜黄色
32〜63	：	黄色〜緑
64〜95	：	緑〜シアン
96〜127	：	シアン〜青
128〜159	：	青〜マゼンダ
160〜191	：	マゼンダ〜赤
192 以上	：	禁止

S は白レベル（0 で白、0x1F で原色）、V は黒レベル（0 で黒、0x1F で原色）を示します。
また RGB はパレットで指定するデータ形式になります（緑・赤・青はそれぞれ 5 ビット。最下位ビットは 0）。

戻り値

RGB を次の形式で返します。

00000000 00000000 GGGGGRRR RRBBBBB0

HSYNCST

レベル 0

書式

```
#include <iocslib.h>

int HSYNCST (ADDRESS) ;
unsigned char *ADDRESS; /* 割り込み処理アドレス */
```

機能

H−SYNC による割り込みを制御します。
ADDRESS には割り込み処理アドレスを指定します（0 ならば割り込み禁止）。

戻り値

割り込みが可能なときは 0、すでに使用中の場合には 0 以外の値を返します。

INDOSFLG

レベル1

書　式　# include <doslib. h>

int INDOSFLG ( ) ;

機　能　OSのワークINDOS_FLGのアドレスを返します。
スーパーバイザ領域なのでユーザーモードではアクセスできないので注意してください。
また、このアドレスが指すワークエリアに続いて、OSにとって重要な情報が格納されているので、読み出すのは構いませんが、絶対に書き直さないでください。

●ライブラリ関数INDOSFLG ( ) の指す内容について

書　式	内　容
unsigned short indosf	OS実行中レベル
unsigned char doscmd	OS実行中ファンクション番号
unsigned char fat_flg	FAT検索モード　0＝標準 0≠先頭から
unsigned short retry_count	I/Oリトライカウント　標準で3回
unsigned short retry_time	I/Oリトライ待ち時間　標準100 (1秒)
unsigned short verifyf	ベリファイモード　0＝オフ 0≠オン
unsigned char breakf	ブレークモード　0＝オフ、1＝オン
unsigned char ctrlpf	CTRL+Pモード　0＝オフ 0≠オン
unsigned char reserved	システムで使用
unsigned char wkcurdrv	カレントドライブ　A:＝0

戻り値　INDOS_FLGが格納されているアドレスを返します。

INIT_PRN

レベル0

書式

include <iocslib. h>

int INIT_PRN (LINE, WIDTH) ;
int LINE; /* 印字可能行数 */
int WIDTH; /* 印字可能桁数 */

機能

プリンタポートを初期化します。
LINEには1ページの行数−1を指定します（OUTPRN関数用。0xFFでページ指定なし)。
WIDTHに1行の桁数−1を指定します（OUTPRN関数用。0xFFで行指定なし)。

戻り値

0x000020ならばプリンタへの出力は可能です。
0x000000のときには出力できません。

プログラム例

```
#include        <stdio.h>
#include        <stdlib.h>
#include        <iocslib.h>

#define MAXLINE         66      /* 印字可能行数 */
#define MAXWIDTH        132     /* 印字可能文字数 */

main()
{
        char    *str = "hello¥r¥n";

        if (INIT_PRN(MAXLINE - 1, MAXWIDTH - 1) == 0) {
                printf("プリンタへの出力はできません¥n");
                exit(1);
        }
        if (SNSPRN() == 0) {
                printf("プリンタへの出力はできません¥n");
                exit(1);
        }
        while (*str)
                OUTLPT(*str++);
}
```

INKEY

レベル 1

書　式

```
#include <doslib.h>

int INKEY ( ) ;
```

機　能

キーが押下されるまで待ち、文字コードを入力します。
ブレークチェックは行いません。

戻り値

入力された文字コードを返します。

参照関数

GETC, GETCHAR

INP232C

レベル 0

書　式　# include <iocslib. h>

int INP232C ( ) ;

機　能　RS-232C 受信データを獲得します。

戻り値　RS-232C 受信データを 1 バイト返します。

プログラム例

```
#include        <iocslib.h>

main()
{
        int     data;           /* 受信データ */

        if ((data = ISNS232C()) != 0)
                data = INP232C();
}
```

INPOUT

レベル 1

書式

```
#include <doslib.h>

int INPOUT (code) ;
int code;  /*  ファンクションコード  */
```

機能

コンソールの直接入出力を行います。
codeが0xFFならばキー入力を行います。
このとき、入力待ちは行いません。
codeが0xFEならばキーセンスを行い、入力したキーコードを返します。
これ以外の場合には、codeを文字コードとして出力します。
なお、ブレークチェックは行いません。

戻り値

0、あるいは入力した文字コードを返します。
文字を出力した場合には必ず0を返します。
入力時に戻り値が0ならば、入力がなかったことを示します。

参照関数

GETCHAR, PUTCHAR, GETC, INKEY

INTVCG

レベル 1

書式

#include <doslib. h>

int INTVCG (intno) ;
int intno ; /* 割り込みベクタ番号 */

機能

intnoで指定する割り込みベクタの処理アドレスを調べます。
なお、割り込みベクタ0x0~0xFFは割り込み処理、割り込みベクタ0x100~0x1FFはIOCSコール、割り込みベクタ0xFF00~0xFFFFはOSのファンクションコールに割りあてられています。

戻り値

割り込みベクタの処理アドレスを返します。

参照関数

INTVCS

プログラム例

```
#include        <stdio.h>
#include        <doslib.h>

#define IOCS            0x0100
#define FUNC            0xff00

main()
{
        int     intno;          /* ベクタ番号 */

        intno = IOCS;           /* IOCS */

        printf("0x0100: 0x%x¥n", INTVCG(intno));

        intno = FUNC;           /* ファンクションコール */

        printf("0xff00: 0x%x¥n", INTVCG(intno));
}
```

INTVCS

レベル 1

書　式

#include <doslib. h>

int INTVCS (intno, jobadr) ;
int intno ;　/*　ベクタの番号　*/
char *jobadr ;　/*　処理ルーチンのエントリアドレス　*/

機　能

intnoで指定する割り込みベクタに、jobadrで示す処理ルーチンの処理アドレスを設定します。
intnoに指定できるベクタは次の通りです。

0x00～0xFF ： 割り込み処理
0x100～0x1FF ： IOCSコール
0xFF00～0xFFFF ： OSのファンクションコール

戻り値

設定変更前のベクタの処理アドレスを返します。

参照関数

INTVCG

プログラム例

```
#include        <stdio.h>
#include        <doslib.h>

#define INR     0x6c
#define INRADR  0xa000

main()
{
        int     intno;          /* ベクタ番号 */
        char    *jobadr;        /* 処理ルーチンエントリ */

        intno = INR;
        jobadr = (char *)INRADR;

        INTVCS(intno, jobadr);  /* ベクタ設定 */

        printf("0x%x: 0x%x¥n", intno, INTVCG(intno));
}
```

IOCTRLDVCTL　レベル 1

書　式

```
# include <doslib. h>

int IOCTRLDVCTL (drive, f_code, buffer) ;
int drive ; /* ドライブ番号 */
int f_code ; /* ファンクション番号 */
unsigned char * buffer ; /* 情報格納領域へのポインタ */
```

機　能

ドライブ番号 drive とファンクション番号 f_code を指定して、デバイスドライバによる特殊コントロールを行います。
buffer には、デバイスドライバとの情報の受け渡しを行うための領域を指定します。
必要となる領域の大きさはデバイスドライバにより異なります。
処理内容はデバイスドライバにより異なります。

戻り値

戻り値は、デバイスドライバにより異なります。
負の数ならエラーを表します。

参照関数

IOCTRLDVGT, IOCTRLFDCTL, IOCTRLFDGT, IOCTRLGT, IOCTRLIS, IOCTRLOS, IOCTRLRD, IOCTRLRH, IOCTRLRTSET, IOCTRLST, IOCTRLWD, IOCTRLWH

IOCTRLDVGT

レベル 1

書　　式

```
# include <doslib. h>

int IOCTRLDVGT (drive) ;
int drive ;  /*  調べたいドライブ番号  */
```

機　　能　指定されたドライブがローカルかリモートかを調べます。

戻 り 値　指定されたドライブがローカルならば、デバイスドライバのデバイス情報を返します。
リモートならば、戻り値のビット 12 が 1 になります。
負の数ならエラーを表します。

参照関数　IOCTRLDVCTL, IOCTRLFDCTL, IOCTRLFDGT, IOCTRLGT, IOCTRLIS, IOCTRLOS, IOCTRLRD, IOCTRLRH, IOCTRLRTSET, IOCTRLST, IOCTRLWD, IOCTRLWH

IOCTRLFDCTL　レベル 1

書　式

```
# include <doslib. h>

int IOCTRLFDCTL (fineno, f_code, buffer) ;
int fileno ; /* ファイルハンドル */
int f_code ; /* ファンクション番号 */
unsigned char * buffer ; /* 情報格納領域へのポインタ */
```

機　能

ファイルハンドル fileno とファンクションコード f_code を指定して、デバイスドライバによる特殊コントロールを行います。
buffer は、デバイスドライバとの情報の受け渡しを行うための領域を指定します。
必要となる領域の大きさはデバイスドライバにより異なります。
処理内容はデバイスドライバにより異なります。

戻り値

戻り値は、デバイスドライバにより異なります。
負の数ならエラーを表します。

参照関数

IOCTRLDVCTL, IOCTRLDVGT, IOCTRLFDGT, IOCTRLGT, IOCTRLIS, IOCTRLOS, IOCTRLRD, IOCTRLRH, IOCTRLRTSET, IOCTRLST, IOCTRLWD, IOCTRLWH

IOCTRLFDGT

レベル 1

書　式

```
# include <doslib.h>

int IOCTRLFDGT (fileno) ;
unsigned fileno ; /* 調べたいファイルのファイルハンドル */
```

機　能

指定されたファイルがローカルかリモートかを返します。

戻り値

指定されたファイルがローカルならば、デバイスドライバのデバイス情報を返します。
リモートならば、戻り値のビット 12 が 1 になります。
負の数ならエラーを表します。

参照関数

IOCTRLDVCTL, IOCTRLDVGT, IOCTRLFDCTL, IOCTRLGT, IOCTRLIS, IOCTRLOS, IOCTRLRD, IOCTRLRH, IOCTRLRTSET, IOCTRLST, IOCTRLWD, IOCTRLWH

IOCTRLGT

レベル 1

書　式　# include <doslib. h>

```
int IOCTRLGT (fileno) ;
int fileno; /* 調べたいファイルのファイルハンドル */
```

機　能　filenoで指定されたファイルハンドルのデバイス情報を調べます。
本関数は、デバイスドライバより直接入出力を行うときに使用します。

戻り値　デバイス情報を返します。
デバイス情報の各ビットが1であれば次の内容を表します。

ビット 0　標準入力デバイス（CON）であることを表す
ビット 1　標準出力デバイス（CON）であることを表す
ビット 2　NULデバイスであることを表す
ビット 3　CLOCKデバイスであることを表す
ビット 5　RAWモードであることを表す（0ならばCOOKEDモード）
ビット 6　特殊IOCTRLが可能であることを表す
このビットが1でないと、IOCTRLDVCTL, IOCTRLFDCTLは使用できない
ビット 7　キャラクタデバイスであることを表す（0ならばブロックデバイスであり、ビット0～4はドライブ番号を表す）
ビット 12　特殊ブロックデバイスの時、デバイスがリモートであることを表す（0ならばローカル）
ただし、ビット13が1の時のみ意味を持つ
ビット 13　特殊ブロックデバイスであることを表す（0ならば通常のブロックデバイス）
ただし、ビット7が0の時のみ意味を持つ
ビット 14　IOCTRLが可能であることを表す
このビットが1でないと、IOCTRLRH, IOCTRLWH, IOCTRLRD, IOCTRLWDは使用できない
その他のビットはシステムで予約

ビット7が0の場合（ブロックデバイスの場合）
ビット0～4の5ビットはドライブ番号（0～25, A: = 0）を表します。
負の数ならエラー表します。

参照関数 IOCTRLDVCTL, IOCTRLDVGT, IOCTRLFDCTL, IOCTRLFDGT, IOCTRLIS, IOCTRLOS, IOCTRLRD, IOCTRLRH, IOCTRLRTSET, IOCTRLST, IOCTRLWD, IOCTRLWH

プログラム例

```
#include        <stdio.h>
#include        <fcntl.h>
#include        <stat.h>
#include        <io.h>
#include        <doslib.h>

main()
{
        int     fno;            /* ファイルハンドル */

        fno = open("A:¥¥AUTOEXEC.BAT", O_RDWR);
        printf("0x%x¥n", IOCTRLGT(fno));
        close(fno);

        fno = open("CON", O_RDWR);
        printf("0x%x¥n", IOCTRLGT(fno));
        close(fno);
}
```

IOCTRLIS

レベル 1

書式 # include <doslib. h>

int IOCTRLIS (fileno) ;
int fileno ; /* 調べたいファイルのファイルハンドル */

機能 指定したファイルハンドルの入力ステータスを調べます。

戻り値 次のいずれかのステータスを返します。

−1 ： 入力可
0 ： 入力不可

参照関数 IOCTRLDVCTL, IOCTRLDVGT, IOCTRLFDCTL, IOCTRLFDGT, IOCTRLGT, IOCTRLOS, IOCTRLRD, IOCTRLRH, IOCTRLRTSET, IOCTRLST, IOCTRLWD, IOCTRLWH

プログラム例

```
#include        <stdio.h>
#include        <fcntl.h>
#include        <stat.h>
#include        <io.h>
#include        <doslib.h>

main()
{
        int     fno;            /* ファイルハンドル */
        int     status;         /* 入力ステータス */

        fno = open("A:¥¥CONFIG.SYS", O_RDONLY);

        status = IOCTRLIS(fno);

        printf("0x%02x¥n", status);
}
```

IOCTRLOS

レベル 1

書式　# include <doslib. h>

int IOCTRLOS (fileno) ;
int fileno ;　/*　調べたいファイルのファイルハンドル　*/

機能　指定したファイルハンドルの出力ステータスを調べます。

戻り値　次のいずれかのステータスを返します。

−1 ： 出力可
0 ： 出力不可

参照関数　IOCTRLDVCTL, IOCTRLDVGT, IOCTRLFDCTL, IOCTRLFDGT, IOCTRLGT, IOCTRLIS, IOCTRLRD, IOCTRLRH, IOCTRLRTSET, IOCTRLST, IOCTRLWD, IOCTRLWH

プログラム例

```
#include        <stdio.h>
#include        <fcntl.h>
#include        <stat.h>
#include        <io.h>
#include        <doslib.h>

main()
{
        int     fno;            /* ファイルハンドル */
        int     status;         /* 出力ステータス */

        fno = open("A:¥¥CONFIG.SYS", O_RDONLY);

        status = IOCTRLOS(fno);

        printf("0x%02x¥n", status);
}
```

IOCTRLRD

レベル 1

書式 #include <doslib.h>

int IOCTRLRD (drive, buffer, size) ;
int drive ; /* ドイラブ番号 */
unsigned char *buffer ; /* 読み込みデータ格納領域へのポインタ */
int size ; /* 読み込みバイト数 */

機能 driveで指定するドライブのデバイスドライバから、sizeで指定するバイト数を、bufferが指す領域へ読み込みます。
IOCTRLRDは、デバイスドライバより直接入力を行うときに使用します。

戻り値 正常終了の場合、読み込んだバイト数を返します。
負の数ならばエラーを表します。

参照関数 IOCTRLDVCTL, IOCTRLDVGT, IOCTRLFDCTL, IOCTRLFDGT, IOCTRLGT, IOCTRLIS, IOCTRLOS, IOCTRLRH, IOCTRLRTSET, IOCTRLST, IOCTRLWD, IOCTRLWH

プログラム例

```
#include        <stdio.h>
#include        <doslib.h>

#define DRIVE           0       /* ドライブ番号 */
#define SIZE            256     /* データbyte数 */

main()
{
        char    buf[SIZE];      /* データ格納領域 */
        int     n;              /* 読んだデータ数 */

        if ((n = IOCTRLRD(DRIVE, buf, SIZE)) >= 0)
                printf("%d byte 読みました。¥n", n);
        else {
                if ((IOCTRLDVGT(DRIVE) & 0x4000) == 0)
                        printf("IOCTRLRDできないデバイス¥n");
                else
                        printf("エラーです。¥n");
        }
}
```

IOCTRLRH

レベル 1

書　式　# include <doslib. h>

int IOCTRLRH (fileno, buffer, size) ;
int fileno ; /* ファイルのファイルハンドル */
unsigned char *buffer ; /* 読み込みデータ格納領域へのポインタ */
int size ; /* 読み込みバイト数 */

機　能　fileno で指定するファイルハンドルのデバイスドライバから、size で指定するバイト数を、buffer が指す領域へ読み込みます。
IOCTRLRH 関数は、デバイスドライバより直接入力を行うときに使用します。

戻り値　正常終了の場合、読み込んだバイト数を返します。
負の数ならエラー表します。

参照関数　IOCTRLDVCTL, IOCTRLDVGT, IOCTRLFDCTL, IOCTRLFDGT, IOCTRLGT, IOCTRLIS, IOCTRLOS, IOCTRLRD, IOCTRLRTSET, IOCTRLST, IOCTRLWD, IOCTRLWH

プログラム例

```
#include        <stdio.h>
#include        <fcntl.h>
#include        <stat.h>
#include        <io.h>
#include        <doslib.h>

#define SIZE    128             /* データbyte数 */

main()
{
        char    buf[SIZE];      /* データ格納領域 */
        int     fno;            /* ファイルハンドル */
        int     n;              /* 読んだデータ数 */

        fno = open("A:¥¥CONFIG.SYS", O_RDONLY);

        if ((n = IOCTRLRH(fno, buf, SIZE)) >= 0)
                printf("%d byte 読みました。¥n", n);
        else {
                if ((IOCTRLFDGT(fno) & 0x4000) == 0)
                        printf("IOCTRLRHできないデバイス¥n");
                else
                        printf("エラーです。¥n");
        }
        close(fno);
}
```

IOCTRLRTSET

レベル 1

書　式

```
# include <doslib. h>

int IOCTRLRTSET (count, time) ;
int count ;  /*  リトライ回数 */
int time ;  /*  時間 */
```

機　能

リトライ数を設定します。
デフォルトの値は3回で、時間は100（1秒）です。
この関数はデバイスドライバでサポートされます。

戻り値

負の数ならエラーを表します。

参照関数

IOCTRLDVCTL, IOCTRLDVGT, IOCTRLFDCTL, IOCTRLFDGT, IOCTRLGT, IOCTRLOS, IOCTRLRD, IOCTRLRH, IOCTRLST, IOCTRLWD, IOCTRLWH

IOCTRLST

レベル 1

書　式　#include <doslib. h>

int IOCTRLST (fileno, data) ;
int fileno ; /*　ファイルのファイルハンドル　*/
int data ; /*　設定するデバイス情報　*/

機　能　filenoで指定されたファイルハンドルにデバイス情報を設定します。
filenoは、キャラクタデバイスに対してオープンされたファイルハンドルでなければなりません。
設定できるのは、ビット5のRAWモード設定のみで、次の意味をもちます。

0x20 ： RAWモード指定
0x00 ： COOKEDモード指定

戻り値　変更後のデバイス情報を返します。
負の数ならエラーを表します。

参照関数　IOCTRLDVCTL, IOCTRLDVGT, IOCTRLFDCTL, IOCTRLFDGT, IOCTRLGT, IOCTRLIS, IOCTRLOS, IOCTRLRD, IOCTRLRH, IOCTRLRTSET, IOCTRLWD, IOCTRLWH

プログラム例

```
#include        <stdio.h>
#include        <fcntl.h>
#include        <stat.h>
#include        <io.h>
#include        <doslib.h>

#define RAW             0x20
#define COOKED          0x00

main()
{
        int     fno;    /* ファイルハンドル */

        fno = open("CON", O_RDONLY);
        printf("old No. = %04x¥n", IOCTRLGT(fno));

        IOCTRLST(fno, RAW);
        printf("new No. = %04x¥n", IOCTRLGT(fno));

        IOCTRLST(fno, COOKED);
        printf("new No. = %04x¥n", IOCTRLGT(fno));

        close(fno);
}
```

IOCTRLWD

レベル 1

書　式　#include <doslib. h>

int IOCTRLWD (drive, buffer, size) ;
int drive ; /＊　ドライブ番号　＊/
unsigned char ＊buffer ; /＊　書き出しデータ格納領域へのポインタ　＊/
int size ; /＊　書き出しバイト数　＊/

機　能　bufferが指す領域から、sizeで指定するバイト数のデータを、driveで指定するドライブのデバイスドライバに書き出します。
IOCTRLWD関数は、デバイスドライバに直接出力を行うときに使用します。

戻り値　正常終了の場合、書き込んだバイト数を返します。
負の数ならエラーを表します。

参照関数　IOCTRLDVCTL, IOCTRLDVGT, IOCTRLFDCTL, IOCTRLFDGT, IOCTRLGT, IOCTRLIS, IOCTRLOS, IOCTRLRD, IOCTRLRH, IOCTRLRTSET, IOCTRLST, IOCTRLWH

プログラム例

```
#include        <stdio.h>
#include        <doslib.h>

#define DRIVE           0       /* ドライブ番号 */
#define SIZE            256     /* データbyte数 */

main()
{
        char    buf[SIZE];      /* データ格納領域 */
        int     n;              /* 書いたデータ数 */

        if ((n = IOCTRLWD(DRIVE, buf, SIZE)) >= 0)
                printf("%d byte 書きました。¥n", n);
        else {
                if ((IOCTRLDVGT(DRIVE) & 0x4000) == 0)
                        printf("IOCTRLWDできないデバイス¥n");
                else
                        printf("エラーです。¥n");
        }
}
```

IOCTRLWH

レベル 1

書　式

```
#include <doslib.h>

int IOCTRLWH (fileno, buffer, size) ;
int fileno ;  /*  ファイルのファイルハンドル  */
unsigned char *buffer ;  /*  書き出しデータ格納領域へのポインタ  */
int size ;  /*  書き出しバイト数  */
```

機　能

bufferが指す領域から、sizeで指定するバイト数をfilenoで指定するファイルハンドルのデバイスドライバに書き出します。

IOCTRLWH関数は、デバイスドライバに直接出力を行うときに使用します。

戻り値

正常終了の場合、書き込んだバイト数を返します。

負の数ならエラーを表します。

参照関数

IOCTRLDVCTL, IOCTRLDVGT, IOCTRLFDCTL, IOCTRLFDGT, IOCTRLGT, IOCTRLIS, IOCTRLOS, IOCTRLRD, IOCTRLRH, IOCTRLRTSET, IOCTRLST, IOCTRLWD

プログラム例

```
#include        <stdio.h>
#include        <fcntl.h>
#include        <stat.h>
#include        <io.h>
#include        <doslib.h>

#define SIZE            256     /* データbyte数 */

main()
{
        char    buf[SIZE];      /* データ格納領域 */
        int     fno;            /* ファイルハンドル */
        int     n;              /* 書いたデータ数 */

        fno = open("A:¥¥CONFIG.SYS", O_RDWR);

        if ((n = IOCTRLWH(fno, buf, SIZE)) >= 0)
                printf("%d byte 書きました。¥n", n);
        else {
                if ((IOCTRLFDGT(fno) & 0x4000) == 0)
                        printf("IOCTRLWHできないデバイス¥n");
                else
                        printf("エラーです。¥n");
        }
        close(fno);
}
```

IPLERR

レベル 0

書　式

```
#include <iocslib.h>

void IPLERR ( ) ;
```

機　能　IPL プログラムで、ディスクエラーなどのエラーが発生した場合に使用します。

戻り値　リターンしません。

ISNS232C

レベル 0

書　式　#include <iocslib. h>

int ISNS232C ( ) ;

機　能　RS-232C受信データをチェックします。

戻り値　0x10000＋受信データを返します。
戻り値が0のときは、受信データがないことを示します。

プログラム例

```
#include        <stdio.h>
#include        <iocslib.h>

main()
{
        int     data;   /* 受信データ */

        data = ISNS232C();

        if (data == 0)
                puts("受信データがありません。");
}
```

JISSFT

レベル 0

書式

#include <iocslib. h>

int JISSFT (CODE) ;
int CODE; /* JIS 漢字コード */

機能

JIS 漢字コードをシフト JIS コードに変換します。

戻り値

シフト JIS コードを返します。
ただし、上位 16 ビットが 0xFFFF ならエラーが発生したことを示します。

プログラム例

```
#include        <stdio.h>
#include        <iocslib.h>

main()
{
        int     jis;    /* JIS漢字コード */
        int     ms;     /* シフトJISコード */

        jis = 0x4f53;

        ms = JISSFT(jis);

        printf("0x%04x(jis) : 0x%04x(sft)¥n", jis, ms);
}
```

JOYGET

レベル0

書式

```
#include <iocslib.h>

int JOYGET (STICKNO) ;
int STICKNO /*  ジョイスティック番号  */
```

機能

ジョイスティックのデータを読み込みます。
ビットが0ならばONであることを示します。
STICKNOにはジョイスティック番号を指定します（0ならばジョイスティック1、1ならばジョイスティック2となります）。

戻り値

次の値を返します。

1	S2	S1	1	R	L	D	U
	Bボタン	Aボタン		右	左	下	上

KEEPPR

レベル 1

書　式　#include <doslib. h>

void KEEPPR (prglen, code) ;
int prglen ; /*　プロセスの常駐バイト数　*/
int code ; /*　終了コード　*/

機　能　プロセスの総バイト数のうち、prglenで指定されたバイト数（PSP部分を除く）の部分を常駐させたまま、プロセスを終了させます。
現在オープンされているファイルハンドルは、すべてクローズされます。
子プロセスがオープンしたファイルハンドルについても同じです。
prglen は_HEND－_PSP－0xF0 で求めることができます。
_HEND と_PSP は int の外部変数です。

戻り値　戻り値はありません。

参照関数　EXIT, EXIT2

プログラム例

```
#include        <doslib.h>

extern int      _HEND;
extern int      _PSP;

main()
{
        int     code;           /* 終了コード */

        code = 0;

        KEEPPR(_HEND - _PSP - 0xf0, code);
}
```

KEYSNS

レベル1

書　式

```
#include <doslib.h>

int KEYSNS ( );
```

機　能

標準入力の入力状態を調べます。
また、このときブレークチェックも行います（^C：^P：^N：)。

戻り値

入力があるときには−1、ないときは0を返します。

KFLUSHGC

レベル 1

書　式	# include <doslib. h> int KFLUSHGC ( ) ;
機　能	入力バッファをフラッシュしてから標準入力から 1 文字読み込みます。 このときブレークチェックも行います (^C : ^P : ^N :)。 KFLUSHGC 関数は入力バッファをフラッシュして GETC 関数と同様な処理を行います。
戻り値	読み込んだ文字コードを返します。
参照関数	GETC, KFLUSHGP, KFLUSHIO, KFLUSHIN, KFLUSHGS

KFLUSHGP

レベル 1

書　式

```
#include <doslib.h>

int KFLUSHGP ( ) ;
```

機　能

入力バッファをフラッシュしてから標準入力から1文字読み込み、読み込んだ文字を標準出力にエコーバックします。
このときブレークチェックも行います（^C：^P：^N：）。
KFLUSHGP 関数は入力バッファをフラッシュして GETCHAR 関数と同様な処理を行います。

戻り値

読み込んだ文字コードを返します。

参照関数

GETCHAR, KFLUSHIO, KFLUSHIN, KFLUSHGC, KFLUSHGS

KFLUSHGS

レベル 1

書式

```
#include <doslib.h>

int KFLUSHGS (inpptr) ;
struct INPPTR {  /*  入力文字列格納領域  */
  unsigned char max;  /*  入力時の最大文字数  */
  unsigned char length;  /*  入力が行われた文字数  */
  unsigned char buffer [256] ;  /*  入力データ  */
} * inpptr;
```

機能

入力バッファをフラッシュしてからCRコードが入力データに現れるまで、文字列を入力します。

このときブレークチェックも行います（^C：^P：^N：)。

入力が最大文字数inpmaxを超えた場合は、警告音を出しますが終了はしません。

lengthには実際に入力された文字数が、bufferには入力された文字列データが格納されます。

KFLUSHGS関数は入力バッファをフラッシュしてGETS関数と同様な処理を行います。

戻り値

CRコードを除く入力文字数を返します。

参照関数

GETS, KFLUSHGP, KFLUSHIO, KFLUSHIN, KFLUSHGC

KFLUSHIN

レベル 1

書　式

```
#include <doslib.h>

int KFLUSHIN ( ) ;
```

機　能

キーバッファをフラッシュしてからキーが押下されるまで待ち、文字コードを入力します。
ブレークチェックは行いません。
KFLUSHIN 関数はキーバッファをフラッシュして、INKEY 関数と同様な処理を行います。

戻り値

入力された文字コードを返します。

参照関数

INKEY, KFLUSHGP, KFLUSHIO, KFLUSHGC, KFLUSHGS

KFLUSHIO

レベル 1

書式

```
#include <doslib.h>

int KFLUSHIO (code) ;
int code;  /*  ファンクションコード  */
```

機能

キーバッファをフラッシュしてから、コンソールの直接入出力を行います。
code が 0xFF ならばキー入力を行い、入力がない場合は待つことはしません。
code が 0xFE ならばキーのセンスを行い、入力した文字コードを返します。
これ以外のときは、code を文字コードとして出力します。
なお、ブレークチェックは行いません。
KFLUSHIO 関数はキーバッファをフラッシュして INPOUT 関数と同様な処理を行います。

戻り値

0、あるいは入力した文字コードを返します。
文字を出力した場合には必ず 0 を返します。
入力時に戻り値が 0 ならば、入力がなかったことを示します。

参照関数

INPOUT, KFLUSHGP, KFLUSHIN, KFLUSHGC, KFLUSHGS

KILL_PR

レベル 1

書　式　# include <doslib. h>

```
int KILL_PR ( ) ;
```

機　能　自分自身のプロセスを削除します。
常駐終了していたプロセスの場合、同じ psp_id を持つスレッドを全て削除し、確保されていたメモリを開放します。
KILL_PR 関数を実行する時は、自分でオープンしたファイルは全てクローズし、書き換えたベクタなども元に戻した後にしてください。
システムは、スレッドの削除とメモリの開放しか行いません。
プロセス内でいくつかのスレッドを登録し実行した後で、EXIT 関数や EXIT2 関数でプロセスを終了する場合は、プロセス内でオープンしたスレッドのみを KILL_PR 関数で削除してから、プロセスを終了してください。
メインのスレッドであるプロセスを KILL_PR 関数した場合、その後の動作は保証されません。

戻り値　負の数ならエラーを表します。

参照関数　OPEN_PR, GET_PR, SUSPEND_PR, SEND_PR, TIME_PR, CHANGE_PR

K_INSMOD

レベル 1

書　　式　#include <doslib. h>

void K_INSMOD (n) ;
int n;　/*　INS キーのモード　*/

機　　能　INS キーのモードを切り替えます。
n に 0xFF を指定したときには ON、0x00 を指定したときには OFF となります。

戻 り 値　戻り値はありません。

参照関数　K_KEYINP, K_KEYSNS, K_SFTSNS, K_KEYBIT

プログラム例

```
#include        <stdio.h>
#include        <doslib.h>

#define INS_ON          0xff
#define INS_OFF         0x00

main()
{
        K_INSMOD(INS_ON);       /* 挿入モード */
        puts("挿入モードです。");

        puts("Hit any key");
        while (K_KEYSNS() == 0)
                ;

        K_INSMOD(INS_OFF);      /* 非挿入モード */
        puts("非挿入モードです。");
}
```

K_KEYBIT

レベル 1

書式

```
#include <doslib.h>

int K_KEYBIT (n) ;
int n;  /*  キーコードグループ  */
```

機能 次に示すキーコードグループを指定して、キーの押下状態を調べます。

キーコードグループ＼ビット	0	1	2	3	4	5	6	7
0	未定義	ESC	1 !	2 ”	3 #	4$	5%	6&
1	7 ’	8 (	9)	0	— =	^	¥ |	BS
2	TAB	Q	W	E	R	T	Y	U
3	I	O	P	@	[	CR	A	S
4	D	F	G	H	J	K	L	;
5	:	]	Z	X	C	V	B	N
6	M	, <	.>	/ ?	_	SP	HOME	DEL
7	Roll up	Roll down	UNDO	←	↑	→	↓	CLR
8	/	*	—	7	8	9	+	4
9	5	6	=	1	2	3	ENTER	0
A	,	.	記号	登録	HELP	XF1	XF2	XF3
B	XF4	XF5	かな	ローマ字	コード	CAPS	INS	ひらがな
C	全角	BREAK	COPY	F・1	F・2	F・3	F・4	F・5
D	F・6	F・7	F・8	F・9	F・10	未定義	未定義	未定義
E	SHIFT	CTRL	OPT.1	OPT.2	未定義	未定義	未定義	未定義
F	未定義	未定義	未定義	未定義	未定義	未定義	未定義	未定義

このファンクションコールは、CON デバイスによりサポートされます。

戻り値 上記の表に示す各ビットが0か1かにより、各キーコードグループのキーの押下状態を返します。

ビットが1になっていれば、キーが押されている状態です。

参照関数 K_KEYINP, K_KEYSNS, K_SFTSNS, K_INSMOD

プログラム例

```
#include        <stdio.h>
#include        <doslib.h>

main()
{
        int     keygrp;         /* グループ */
        int     keystas;        /* グループのキー状態 */


        keygrp = 0x0b;

        keystas = K_KEYBIT(keygrp);

        printf("%8b¥n", keystas);
}
```

K_KEYINP

レベル 1

書　式

```
#include <doslib.h>

int K_KEYINP ( ) ;
```

機　能

キーボードからの入力を行います。
このファンクションコールは、CONデバイスによりサポートされます。

戻り値

入力した文字コードを返します。

参照関数

K_KEYSNS, K_SFTSNS, K_KEYBIT, K_INSMOD

K_KEYSNS

レベル 1

書式 # include <doslib. h>

```
int K_KEYSNS ( ) ;
```

機能 キーボードからの入力を先読みします。
ただし、'¥0'のときは入力が行われなかったことを示します。
このファンクションコールは、CON デバイスによりサポートされます。

戻り値 先読みした文字コードを返します。

参照関数 K_KEYINP, K_SFTSNS, K_KEYBIT, K_INSMOD

K_SFTSNS

レベル 1

書　式

#include <doslib. h>

int K_SFTSNS ( ) ;

機　能

シフトキーの押下状態を調べます。
なお、このファンクションコールは、CON デバイスによりサポートされます。

戻り値

シフトキーの押下状態を返します。
ただし、LED の付いているキーは、LED の点灯状態を返します。
ビット 10～ビット 0 の内容は次の通りです。

10	9	8	7	6	5	4	3	2	1	0
全角	ひらがな	INS	CAPS	コード	ローマ	かな	opt2	opt1	ctrl	shift

(いずれもビットが1のとき、押下状態であることを示します)。

参照関数

K_KEYINP, K_KEYSNS, K_KEYBIT, K_INSMOD

プログラム例

```
#include        <stdio.h>
#include        <doslib.h>

main()
{
        int     shtstas;        /* シフトキー状態 */

        shtstas = K_SFTSNS();
        printf("シフトキー: %11b¥n", shtstas);
}
```

LEDMOD

レベル0

書式

```
#include <iocslib.h>

void LEDMOD (KEYCODE, ONOFF) ;
int KEYCODE;  /*  キー番号  */
int ONOFF;  /* ON/OFF モード  */
```

機能

LEDキーのモードを設定します。
ON/OFFには1 (ON)、または0 (OFF) を指定します。
KEYCODEには0～6を指定します。
その内容は次の通りです。

6	5	4	3	2	1	0
全角	ひらがな	INS	CAPS	コード入力	ローマ字	かな

戻り値

戻り値はありません。

LINE

レベル0

書　式

```
#include <iocslib. h>

int LINE (lineptr) ;
srtuct LINEPTR  {
  short x1 ; /*  始点のX座標  */
  short y1 ; /*  始点のY座標  */
  short x2 ; /*  終点のX座標  */
  short y2 ; /*  終点のY座標  */
  unsigned short color ; /*  パレットコード  */
  unsigned short linestyle ; /*  ラインスタイル  */
} *lineptr ;
```

機　能　グラフィック画面にラインを引きます。

戻り値　終了コードを返します。

　0 ：　正常終了
−1 ：　グラフィックは使用不可

LOAD

レベル 1

書式

```
#include <doslib.h>

int LOAD (file, comline, envptr) ;
unsigned char *file ; /* プログラムファイル名へのポインタ */
stuct COMLINE {
  unsigned char len ; /* コマンドラインの長さ */
  unsigned char buffer [255] ; /* コマンドライン文字列 */
} *comline ;
unsigned char *envptr ; /* 環境文字列へのポインタ */
```

機能

comlineで指定するコマンドラインと、envptrで指定する環境にもとづいて、fileで指定するプログラムをロードします。
envptrに0を指定すると、呼び出し側のプログラムと同じ環境になります。
fileの最上位8ビットの意味は次の通りです。

0：拡張子（.R/.Z/.X）にしたがいロードする
1：Rタイプのファイルとしてロードする
2：Zタイプのファイルとしてロードする
3：Xタイプのファイルとしてロードする

戻り値

ロードしたプログラムの実行アドレスを返します。
負の数ならエラーを表します。

参照関数

LOADEXEC, PATHCHK, LOADONLY, EXECONLY

LOADEXEC

レベル 1

書　式

```
#include <doslib.h>

int LOADEXEC (file, comline, envptr) ;
unsigned char *file; /*  プログラムファイル名へのポインタ  */
struct COMLINE {
  unsigned char len; /*  コマンドラインの長さ  */
  unsigned char buffer [255] ; /*  コマンドライン文字列  */
} *comline;
unsigned char *envptr; /*  環境文字列へのポインタ  */
```

機　能

comlineで指定するコマンドラインと、envptrで指定する環境にもとづいて、fileで指定するプログラムをロード、実行します。
envptrに0を指定すると、呼び出し側のプログラムと同じ環境になります。
fileの最上位8ビットの意味は次の通りです。

0：拡張子（.R/.Z/.X）にしたがいロードする
1：Rタイプのファイルとしてロードする
2：Zタイプのファイルとしてロードする
3：Xタイプのファイルとしてロードする

戻り値

ロード、実行ができなかった場合にはエラーコード（負の数）を、正常終了した場合にはプロセス終了コード（正の数）を返します。

参照関数

LOAD, PATHCHK, LOADONLY, EXECONLY

LOADONLY

レベル 1

書式

```
#include <doslib.h>

int LOADONLY (file, loadadr, limitadr);
unsigned char *file;  /* ファイル名格納領域へのポインタ */
unsigned char *loadadr;  /* ロードアドレス */
unsigned char *limitadr;  /* リミットアドレス */
```

機能

loadadr, limitadr にそれぞれロードアドレス、リミットアドレスを指定して、file のプログラムをロードします。
file の最上位 8 ビットの意味は次の通りです。

0：拡張子（. R/. Z/. X）にしたがいロードする
1：R タイプのファイルとしてロードする
2：Z タイプのファイルとしてロードする
3：X タイプのファイルとしてロードする

戻り値

プログラムのサイズを返します。
負の数ならエラーを表します。

参照関数

LOADEXEC, LOAD, PATHCHK, EXECONLY

LOCK

レベル 1

書　式　# include <doslib. h>

int LOCK (fileno, offset, len) ;
int fileno ; /*　ファイルハンドル　*/
int offset ; /*　ロックする部分へのオフセット　*/
int len ; /*　ロックする部分の長さ　*/

機　能　ファイルのロックを行い、他のプロセスからのファイルアクセスを禁止します。
fileno は、ロックしたいファイルのファイルハンドルを指定します。
offset には、ファイル中のロックしたい部分の、ファイルの先頭からのオフセットを指定します。
len には、ファイル中のロックしたい部分のバイト数を指定します。
この LOCK 関数と UNLOCK 関数により、ファイルアクセスの排他制御を行い、自分以外のプロセスが勝手にファイルを書き変えるのを防ぐことができます。

戻り値　負の数ならエラーを表します。

参照関数　UNLOCK

プログラム例

```
#include        <stdio.h>
#include        <fcntl.h>
#include        <stat.h>
#include        <io.h>
#include        <doslib.h>

#define SIZE    256

static char     buf[SIZE];


main()
{
        int     fd;

        if ((fd = open("A:¥¥vjeb.dic", O_RDONLY|O_BINARY)) == -1)
                puts("can't open");
        else if (LOCK(fd, 0, SIZE) < 0)
                puts("can't lock");
        else if (read(fd, buf, SIZE) != SIZE)
                puts("can't read");
        if (UNLOCK(fd, 0, SIZE) < 0)
                puts("can't unlock");
        close(fd);
}
```

LOF232C

レベル 0

書式　# include <iocslib. h>

int LOF232C ( ) ;

機能　RS-232C 受信バッファ内のデータ数を調べます。

戻り値　RS-232C 受信バッファ内のデータ数を返します。

プログラム例

```
#include        <stdio.h>
#include        <iocslib.h>

main()
{
        int     n;              /* データ数 */

        n = LOF232C();

        printf("data in buffer = %dbytes¥n", n);

}
```

MAKEASSIGN

レベル１

書　式

```
# include <doslib. h>

int MAKEASSIGN (buff1, buff2, mode) ;
unsigned char *buff1; /* ドライブ名 */
unsigned char *buff2; /* ディレクトリ名 */
int mode; /* 割り当てモード */
```

機　能

仮想ドライブ、仮想ディレクトリの割り当てを行います。
modeの値により、以下のような割り当てを行います。

- modeが0x50のとき（仮想ドライブの割り当て）
  buff1のドライブをアクセスすると、buff2のディレクトリをアクセスします。

- modeが0x60のとき（仮想ディレクトリの割り当て）
  buff2のディレクトリをアクセスすると、buff1のドライブをアクセスします。

戻り値

負の数ならエラーを表します。

参照関数

GETASSIGN, RASSIGN

MAKETMP

レベル 1

書　式

```
# include <doslib. h>

int MAKETMP (file, atr) ;
unsigned char *file ; /* パス名格納領城へのポインタ */
int atr ; /* ファイル属性 */
```

機　能

指定したパスにテンポラリファイルを作成します。
file には、テンポラリファイルを作成するディレクトリを示すパス名を設定し、続けて次に示すようなファイル名を設定します。

file????.txt

????の部分は、0000～9999 までの数字に変換されます。
????の代わりに数字を設定すると、その数から検索され、ファイル名が決定します。
atr には、作成するテンポラリファイルの属性を指定します。
ファイル属性の指定方法は CREATE 関数と同じなので、そちらを参照してください。

戻り値

作成したテンポラリファイルのファイルハンドルを返します。
負の数ならエラーを表します。

参照関数

CREATE

プログラム例

```
#include        <stdio.h>
#include        <stdlib.h>
#include        <string.h>
#include        <doslib.h>

static char     *template = "stm0000";
static char     tmp[256];


main()
{
        int     fd1;
        int     fd2;

        strcpy(tmp, template);
        if ((fd1 = MAKETMP(tmp, 0x20)) < 0)
                fprintf(stderr, "can't open temp file¥n");
        strcpy(tmp, template);
        if ((fd2 = MAKETMP(tmp, 0x20)) < 0)
                fprintf(stderr, "can't open temp file¥n");

        WRITE(fd1, "hello", 6);
        WRITE(fd2, "hello", 6);

        CLOSE(fd2);
        CLOSE(fd1);

        system("dir");
        exit(0);
}
```

MALLOC

レベル 1

書　式	# include <doslib. h> int MALLOC (size) ; int size ; /＊　確保する領域のサイズ　＊/
機　能	sizeで指定されたバイト数のメモリ領域を確保し、そのアドレスを返します。 size に 0x1000000 以上の値を指定したときは、0xFFFFFF として処理します。
戻り値	メモリが確保できたときは確保した領域を指すアドレスを返します。 確保できなかったときには 0x81000000＋最大バイト数を返します。 0x8200000?は、まったく確保できないことを示します。

MALLOC2

レベル 1

書　式

```
# include <doslib. h>

int MALLOC2 (md, size) ;
int md ; /*  メモリ割り当てのモード */
int size ; /*  確保したい領域のサイズ */
```

機　能

sizeで指定されたバイト数のメモリ領域を確保し、そのポインタを返します。
mdには、以下に示すメモリの割り当て方法を指定します。

0：下位から探し、割り当てる。
1：指定されたサイズを満たすメモリブロックのうち最小ブロックを割り当てる。
2：上位の方から探し、割り当てる。

sizeに0x1000000以上の値を指定した時は、0xFFFFFFとして処理します。

戻り値

メモリが確保できた時は、その領域を指すポインタを返します。
確保できなかった時は、0x81000000＋最大バイト数を返します。
0x8200000?はまったく確保できないことを表します。

参照関数

MALLOC, MFREE

MFREE

レベル 1

書　式

```
#include <doslib.h>

int MFREE (memptr);
int memptr;  /* メモリブロックを指すアドレス */
```

機　能

memptrで指定したメモリブロックを解放します。
memptrはMALLOCが返した値です。
したがって、誤ったmemptrを指定するとエラーになります。
memptrに0を指定すると、確保したメモリ領域をすべて解放します。
また、子プロセスが確保した領域についても同様です。

戻り値

正の数（0も）で正常終了、負の数（−1以外も）でエラー。

MKDIR

レベル 1

書式

```
#include <doslib.h>

int MKDIR (file) ;
unsigned char *file;  /*  ディレクトリ名格納領域へのポインタ  */
```

機能

ディレクトリを作成します。

戻り値

正の数（0も）で正常終了、負の数（−1以外も）でエラー。
通常は0が返りますが、特殊デバイスドライバが対象の時は、0以外の正の数の場合があります。

参照関数

RMDIR, CHDIR

MOVE

レベル 1

書式

```
#include <doslib.h>

int MOVE (old, new) ;
unsigned char *old; /* 旧ファイル名へのポインタ */
unsigned char *new; /* 新ファイル名へのポインタ */
```

機能

oldで指定したファイルの名前を、newで示す名前に変更します。
oldとnewのアクセスパスが異なる場合は、ディレクトリ間の移動も行います。
ただし、ドライブが異なる場合には、移動はできません。
変更の対象がディレクトリの場合、名前は変更はできますが、移動はできません。
オープンされているファイルに対して実行した場合はエラーになります。

戻り値

正の数（0も）で正常終了、負の数（−1以外も）でエラー。
通常は0が返るが、特殊デバイスドライバが対象の時は、0以外の正の数の場合がある。

参照関数

RENAME

プログラム例

```
#include        <fcntl.h>
#include        <stat.h>
#include        <io.h>
#include        <doslib.h>


main()
{
        int     exist(char *);
        char    *old;
        char    *new;

        old = "A:¥¥CONFIG.SYS";
        new = "A:¥¥CONFIG.BAK";

        if (!exist(old))
                MOVE(new, old);         /* 変更(2回目) */
        else
                MOVE(old, new);         /* 変更(1回目) */
}


static int      exist(char *file)
{
        int     fd;

        if ((fd = open(file, O_RDONLY)) < 0)
                return (0);
        else {
                close(fd);
                return (1);
        }
}
```

MS_CURGT

レベル 0

書　式　# include <iocslib. h>

```
int MS_CURGT ( ) ;
```

機　能　マウスカーソルの座標を調べます。

戻り値　マウスカーソルの座標を返します。
（上位 16 ビットに X 座標を、下位 16 ビットに Y 座標を設定して返します。）

XXXXXXXX XXXXXXXX	YYYYYYYY YYYYYYYY
X 座標	Y 座標

MS_CUROF

レベル0

書　式

```
#include <iocslib.h>

void MS_CUROF ( ) ;
```

機　能　マウスカーソルを消します。

戻り値　戻り値はありません。

MS_CURON

レベル 0

書式 # include <iocslib. h>

```
void MS_CURON ( ) ;
```

機能 マウスカーソルの表示を行います。

戻り値 戻り値はありません。

MS_CURST

レベル0

書　式

```
#include <iocslib.h>

int MS_CURST (X, Y) ;
int X;  /*  X座標  */
int Y;  /*  Y座標  */
```

機　能　マウスカーソルの座標を指定します。

戻り値　処理が正常に終了したら0、エラーが発生した場合には−1を返します。

MS_GETDT

レベル 0

書　　式　# include <iocslib. h>

int MS_GETDT ( ) ;

機　　能　マウスの移動量、およびボタンの ON/OFF を調べます。

戻 り 値　マウスの移動量、およびボタンの ON/OFF の状態を返します。

XXXXXXXX YYYYYYYY LLLLLLL RRRRRRR

ビット 31～24	XXXXXXXX	：	X 方向移動量
ビット 23～16	YYYYYYYY	：	Y 方向移動量
ビット 15～8	LLLLLLLL	：	左ボタンの状態 (0xFF = on, 0x00 = off)
ビット 7～0	RRRRRRRR	：	右ボタンの状態 (0xFF = on, 0x00 = off)

MS_INIT

レベル0

書　　式

```
#include <iocslib.h>

void MS_INIT ( ) ;
```

機　　能　マウスの初期化を行います。

戻 り 値　戻り値はありません。

MS_LIMIT

レベル0

書　式　#include <iocslib.h>

```
int MS_LIMIT (XS, YS, XE, YE) ;
int XS; /* 先頭X座標 */
int YS; /* 先頭Y座標 */
int XE; /* 最終X座標 */
int YE; /* 最終Y座標 */
```

機　能　マウスカーソルの移動範囲を指定します。

戻り値　処理が正常に終了したら0、エラーが発生した場合には－1を返します。

MS_OFFTM

レベル０

書　式

```
#include <iocslib.h>

int MS_OFFTM (MODE, MAXTIME) ;
int MODE ; /*  マウスボタンの指定  */
int MAXTIME ; /*  待ち時間の最大値  */
```

機　能

マウスボタンを離すまでの時間を調べます。
MODE にはマウスボタンを指定します。
0のときは左ボタン、−1のときは右ボタンとなります。
MAXTIME には、待ち時間の最大値を指定します。
0を指定すると、いつまでも待ち続けます。

戻り値

待ち時間を返します。(1～65534)。
ドラッグが行われたときには0、待ち時間の最大値を超えたときには65535を返します。

MS_ONTM

レベル 0

書式

```
#include <iocslib.h>

int MS_ONTM (MODE, MAXTIME) ;
int MODE;  /*  マウスボタンの指定  */
int MAXTIME;  /*  待ち時間の最大値  */
```

機能

マウスボタンを押すまでの時間を調べます。
MODE にはマウスボタンを指定します。
0のときは左ボタン、−1のときは右ボタンとなります。
MAXTIME には、待ち時間の最大値を指定します。
0を指定すると、いつまでも待ち続けます。

戻り値

待ち時間を返します（1〜65534）。
ドラッグが行われたときには0、待ち時間の最大値を超えたときには65535を返します。

MS_PATST

レベル 0

書　　式

```
#include <iocslib.h>

void MS_PATST (CURSORNO, ADDRESS) ;
int CURSORNO; /*  カーソル番号  */
struct PATST *ADDRESS; /*  パターンデータ格納アドレス  */
struct PATST {
  short OFFSET X;
  short OFFSET Y;
  short shadow [16] ;
  short pattern [16] ;
} ;
```

機　　能

マウスカーソルのパターンを登録します。
CURSOLNO には、マウスカーソルの番号を指定します。
ADDRESS には、パターンデータ格納領域のアドレスを指定します。

戻り値

戻り値はありません。

プログラム例

```
#include        <iocslib.h>

#define CURSOLNO        1               /* カーソル番号 */
#define TYPE            8
#define FONTSIZE        32
#define CODE            0x8199

main()
{
        struct FNTBUF   *bufp;
        struct PATST    pat;            /* パターンデータ */
        char    *p;
        int     i;

        i = sizeof(struct FNTBUF) + FONTSIZE;
        bufp = (struct FNTBUF *)malloc(i);
        /* shit! */

        FNTGET(TYPE, CODE, bufp);
        pat.OFFSETX = 0;
        pat.OFFSETY = 0;
        p = bufp->buffer;
        for (i = 0; i < 16; i++) {
                pat.shadow[i] = 0;
                pat.pattern[i]  = (*p++ & 0xff) << 8;
                pat.pattern[i] += (*p++ & 0xff) << 0;
        }
        MS_PATST(CURSOLNO, &pat);

        MS_SEL(CURSOLNO);
        free((char *)bufp);
}
```

MS_SEL

レベル 0

書　式

```
#include <iocslib.h>

void MS_SEL (CURSORNO) ;
int CURSORNO; /*  カーソル番号  */
```

機　能

マウスカーソルを選択します。
CURSORNO にはマウスカーソルの番号を指定します。

戻り値

戻り値はありません。

MS_SEL2

レベル 0

書　式

```
#include <iocslib.h>

void MS_SEL2 (TBLADDRESS) ;
unsigned char *TBLADDRESS; /* カーソル番号テーブルアドレス
*/
```

機　能

マウスカーソルをいくつか指定して、アニメーションを作成します。
TBLADDRESS には、カーソル番号テーブルのアドレスを指定します。
カーソル番号の終わりは、short の−1 です。

戻り値

戻り値はありません。

プログラム例

```
#include        <iocslib.h>

#define TYPE            8
#define FONTSIZE        32
#define CODE1           0x217b
#define CODE2           0x217c

main()
{
        struct PATST    pat1;   /* パターンデータ格納アドレス */
        struct PATST    pat2;   /* パターンデータ格納アドレス */
        struct FNTBUF   *buf1;  /* フォントデータ格納アドレス */
        struct FNTBUF   *buf2;  /* フォントデータ格納アドレス */
        short   curno[10];      /* カーソル番号テーブル */
        unsigned char   *p1;
        unsigned char   *p2;
        int     i;

        i = sizeof(struct FNTBUF) + FONTSIZE;
        buf1 = (struct FNTBUF *)malloc(i);
        buf2 = (struct FNTBUF *)malloc(i);
        /* shit ! */

        /* フォントデータの読み込み */
        FNTGET(TYPE, CODE1, buf1):
        FNTGET(TYPE, CODE2, buf2);

        /* パターンデータの設定 */
        p1 = buf1->buffer;
        p2 = buf2->buffer;
        for (i = 0; i < 16; i++) {
                pat1.shadow[i] = pat2.shadow[i] = 0;
                pat1.pattern[i]  = (*p1++ & 0xff) << 8;
                pat1.pattern[i] += (*p1++ & 0xff) << 0;
                pat2.pattern[i]  = (*p2++ & 0xff) << 8;
                pat2.pattern[i] += (*p2++ & 0xff) << 0;
        }
        MS_PATST(1, &pat1);     /* マウスカーソルの定義 */
        MS_PATST(2, &pat2);

        curno[0] = 1;           /* カーソル番号テーブルの設定 */
        curno[1] = 2;
        curno[2] = -1;

        MS_SEL2(curno);         /* アニメーション指定 */

        free(buf1);
        free(buf2);
}
```

MS_STAT

レベル 0

書　　式　#include <iocslib.h>

```
int MS_STAT ( ) ;
```

機　　能　マウスカーソルの表示モードを調べます。

戻 り 値　マウスカーソルの表示モードを返します。

0 ： 消去
65535 ： 表示

NAMECK

レベル 1

書　式

#include <doslib. h>

int NAMECK (file, buffer) ;
unsigned char ＊file ; /＊　ファイル名格納領域へのポインタ　＊/
struct NAMECKBUF ＊buffer ; /＊　解析結果格納領域へのポインタ　＊/

機　能

ファイル名の解析を行い、その結果を buffer に格納します。
buffer の形式は次の通りです。

struct NAMECKBUF {
　unsigned char drive [2] ; /＊　ドライブ　＊/
　unsigned char path [65] ; /　＊　パス名, ルート ＝ '¥', サブ ＝ '¥test¥'　＊/
　unsigned char name [19] ; /＊　ファイル名　＊/
　unsigned char ext [5] ; /＊　拡張子　＊/
} ;

戻 り 値

負の数ならエラーを表します。
0xFF ならばファイル指定なし、0x00 ならワイルド指定なし、これ以外の正の数のときにはワイルド指定ありとなります。

プログラム例

```
#include        <stdio.h>
#include        <doslib.h>

main()
{
        struct NAMECKBUF        inf;    /* 解析情報 */

        NAMECK("A:¥¥CONFIG.SYS", &inf); /* ファイル名解析 */

        printf("drive = %c%c¥n", inf.drive[0], inf.drive[1]);
        printf("path  = %s¥n", inf.path);
        printf("name  = %s¥n", inf.name);
        printf("ext   = %s¥n", inf.ext);
}
```

NAMESTS

レベル 1

書　式　#include <doslib. h>

```
int NAMESTS (file, buffer) ;
unsignet char *file; /* ファイル名格納領域へのポインタ */
struct NAMESTBUF *buffer; /* 解析情報格納領域へのポインタ */
```

機　能　fileが示すファイル名の解析を行い、解析結果をbufferへ格納します。
bufferの形式は次の通りです。

```
struct NAMESTBUF {
  unsigned char flg; /* 0 = ワイルドなし,0xFF = ファイル指定なし */
  unsigned char drive; /* ドライブ番号 (A = 0, B = 1, …) */
  unsigned char path [65] ; / * パス名 (ルート = '¥',サブ =
                                  '¥test¥') */
  unsigned char name1 [8] ; /* ファイル名 (先頭8文字) */
  unsigned char ext [3] ; /* 拡張子 */
  unsigned char name2 [10] ; /* ファイル名 (9文字目以降) */
} ;
```

戻り値　正の数（0も）で正常終了、負の数（−1以外も）でエラー。

プログラム例

```
#include        <stdio.h>
#include        <string.h>
#include        <doslib.h>

#define NAMELEN         8

main()
{
        struct NAMESTBUF        inf;    /* 解析情報 */
        char    name[NAMELEN + 1];
        char    ext[EXTLEN + 1];

        if (NAMESTS("A:¥¥CONFIG.SYS", &inf) == 0) {
                printf("flg  : %d¥n", inf.flg);
                printf("drive: %d¥n", inf.drive);
                printf("path : %s¥n", inf.path);

                strncpy(name, inf.name1, NAMELEN);
                name[NAMELEN] = '¥0';
                strncpy(ext, inf.ext, EXTLEN);
                ext[EXTLEN] = '¥0';

                printf("name1: %s¥n", name);
                printf("ext  : %s¥n", ext);
                printf("name2: %s¥n", inf.name2);
        }
}
```

NEWFILE

レベル 1

書　　式

```
# include <doslib. h>

int NEWFILE (file, atr) ;
unsigned char *file; /* ファイル名格納領域へのポインタ */
int atr; /* ファイル属性 */
```

機　　能

ファイルを新規に作成します。
CREATE 関数と違うのは、指定したファイルがすでに存在していた場合にエラーを返すことだけです。
art には、作成するファイルの属性を指定します。
ファイル属性の指定方法は CREATE 関数と同じなので、そちらを参照してください。

戻 り 値

作成したファイルのファイルハンドルを返します。
負の数ならエラーを表します。
特に、指定したファイルがすでに存在していた場合には、エラー（−80）を返します。

参照関数

CREATE

NFILES

レベル 1

書　式

```
#include <doslib.h>

int NFILES (buffer) ;
struct FILBUF *buffer ; /*  ファイル情報格納領域へのポインタ  */
struct FILBUF {
  unsigned char OS[21] ; /*  Human68k 内部で使用します  */
  unsigned char atr ; /*  ファイル属性  */
  unsigned short time ; /*ファイルの時刻  */
  unsigned short date ; /*ファイルの日付  */
  unsigned int filelen ; /*ファイルの長さ  */
  unsigned char name[23] ; /*ファイル名  */
} ;
```

機　能

FILESで得たbufferを指定して、次のファイルをサーチします。

戻り値

正の数（0も）で正常終了、負の数（－1以外も）でエラーを表します。

参照関数

FILES

プログラム例

```
#include        <stdio.h>
#include        <doslib.h>

main()
{
        struct FILBUF   inf;      /* ファイル情報 */
        int     atr;              /* ファイル属性 */

        atr = 0x20;               /* 通常ファイル */

        if (FILES(&inf, "*.*", atr) >= 0) {  /* 1つめがあれば */
                printf("%s¥n", inf.name);      /* それを表示 */

                while (NFILES(&inf) >= 0)/* 2つめ以降がある間 */
                        printf("%s¥n", inf.name); /* 表示する */
        }
}
```

ONTIME

レベル 0

書　　式　#include <iocslib. h>

int ONTIME ( ) ;

機　　能　電源投入、またはリセットしてから現在までの時間を返します。

戻 り 値　0秒～23時59分59秒99までの値 (0～8639999) を返します。
この値は1日を超えるとリセットされます。

プログラム例

```
#include        <stdio.h>
#include        <iocslib.h>

main()
{
        int     bin;            /* 時間(2進数) */
        int     h, m, s, ss;

        bin = ONTIME();         /* BIOS動作時間の通知 */

        h = bin / 3600 / 100;
        m = (bin - (h * 3600 * 100)) / 6000;
        s = (bin - (h * 3600 * 100) - (m * 6000)) / 100;
        ss = bin % 100;

        printf("ontime = %d:%d:%d.%d¥n", h, m, s, ss);
}
```

OPEN

レベル 1

書式

```
#include <doslib.h>

int OPEN (file, mode) ;
unsigned char *file;  /*  ファイル名格納領域へのポインタ  */
int mode;  /*  アクセスモード  */
```

機能

fileで指定したファイルをオープンします。
modeには、次のアクセスモードを指定します。

0x000 ： 読み込み
0x001 ： 書き込み
0x002 ： 読み込み/ 書き込み
0x100 ： 読み込み
(辞書用の特殊ハンドルを返す。ユーザーは使用禁止)
0x101 ： 書き込み
(辞書用の特殊ハンドルを返す。ユーザーは使用禁止)
0x102 ： 読み込み/ 書き込み
(辞書用の特殊ハンドルを返す。ユーザーは使用禁止)

戻り値

ファイルハンドルを返します。
負の数ならエラーを表します。

OPEN_PR　レベル1

書　式

```
# include <doslib. h>

int OPEN_PR (name, counter, usp, ssp, sr, pc, buff, sleep_time) ;
unsigned char *name ; /* スレッドの名前へのポインタ */
int counter ; /* タスクの実行間隔の制御パラメータ */
int usp ; /* タスク起動時のuspの初期値 */
int ssp ; /* タスク起動時のsspの初期値 */
int sr ; /* タスク起動時のsrの初期値 */
int pc ; /* タスク起動時の実行開始アドレス */
struct PRCCTRL { /* タスク間通信のための領域 */
  long length ; /* データバッファのバイト数 */
  unsigned char * buf_ptr ; /* データバッファへのポインタ */
  unsigned short command ; /* コマンドバッファ */
  unsigned short your_id ; /* 相手のIDのバッファ
                             (－1で通信許可) */
} * buff ;
long sleep_time ; /* 待ち時間 */
```

機　能

バックグラウンドタスクを登録します。

登録したスレッドはSLEEP状態になります。

nameには、スレッドの名前を表す15文字以内の文字列へのポインタを指定します。

もし、すでに同じ名前のスレッドが起動していたらエラーとなります。

counterには、タスクを1回実行するために、タスク切り替えのためのタイマー割り込みの発生を何回カウントするか、その回数を指定します。

counterの値は2～255です。

0または1が指定された場合は2として扱います。

usp, ssp, sr, pcは、それぞれタスクを実行するときのレジスタの初期値です。

システム用のスタックは6キロバイト必要です。

また、srは0または0x2000を指定し、ユーザーモードかスーパーバイザーモードかどちらかを選択します。

その他のレジスタの初期値は全て0です。

buffには、タスク間通信用のバッファへのポインタを指定します。

sleep_timeには、起動するまでの待ち時間をミリ秒単位で指定します。

0を指定すると、永久にSLEEPします。

OPEN_PR 関数により、バックグラウンドタスクを登録した場合には、KEEP_PR 関数で常駐終了します。
スレッドをメモリ上から消去したい場合は KILL_PR 関数を使用します。

戻り値

スレッドが登録できたならば、スレッドの ID を返します。
負の数ならエラーを表します。
同じ名前のスレッドがすでに起動していた場合は－27 を、これ以上スレッドを起動できない場合は－29 を返します。

参照関数

KILL_PR, GET_PR, SUSPEND_PR, SEND_PR, TIME_PR, CHANGE_PR

OPMINTST

レベル 0

書式

```
#include <iocslib.h>

int OPMINTST (ADDRESS);
unsigned char *ADDRESS; /* 割り込み処理アドレス */
```

機能

FM 音源 IC による割り込みを設定します（MFP のみ制御します)。
ADDRESS には割り込み処理アドレスを指定します。
0 を指定すると割り込み禁止にします。
なお、FM 音源デバイスドライバを組み込んでいる場合は、割り込みを変更してはいけません。

戻り値

割り込みが設定されたときは 0、すでに使用中の場合には 0 以外の値を返します。

OPMSET

レベル０

書　　式

```
#include <iocslib.h>

void OPMSET (ADDRESS, DATA) ;
int ADDRESS ; /*  出力アドレス  */
int DATA ; /*  書き込むデータ  */
```

機　　能

FM音源（YM2151）にデータを書き込みます。
書き込みはFM音源内部のビジーフラグの状態に従って行われます。
なお、FM音源デバイスドライバが曲を演奏している時は、データを書き込んではいけません。

戻り値

戻り値はありません。

OPMSNS

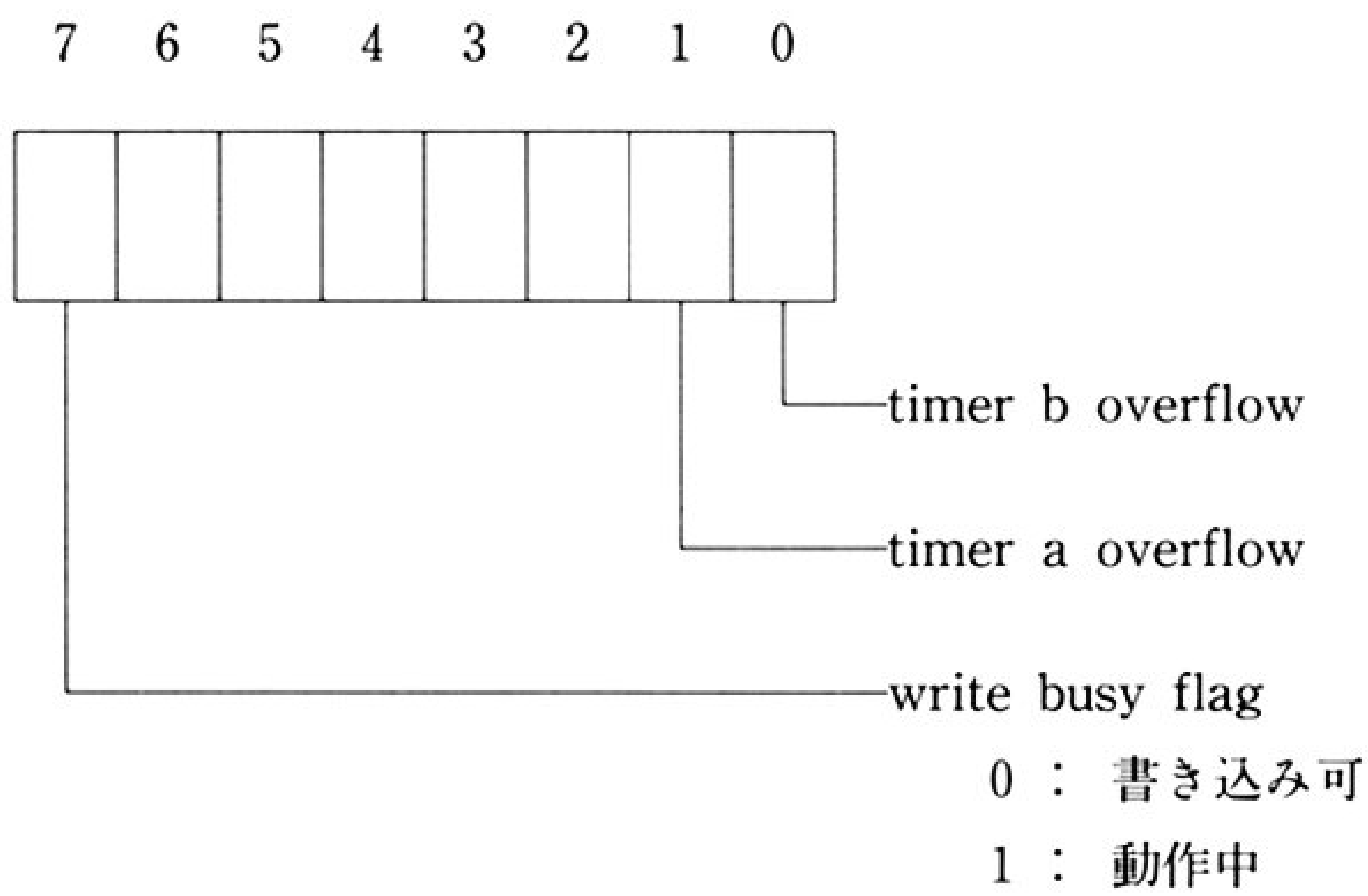
レベル0

書式 # include 〈iocslib. h〉

int OPMSNS ( ) ;

機能 FM 音源（YM2151）のステータスを読み込みます。

戻り値 FM 音源（YM2151）のステータスを返します。



OS_CUROF

レベル 0

書式　#include <iocslib. h>

void OS_CUROF ( ) ;

機能　カーソルを消します（esc[>5h を出力したのと同じ)。

戻り値　戻り値はありません。

OS_CURON

レベル0

書式	# include <iocslib. h> void OS_CURON ( ) ;
機能	カーソルを表示します (esc[>5l を出力したのと同じ)。
戻り値	戻り値はありません。

OSNS232C

レベル 0

書　式　#include <iocslib. h>

int OSNS232C ( ) ;

機　能　RS-232C送信が可能か否かチエックします。

戻り値　戻り値が0のときは送信できません。
戻り値が0以外のときは送信できる（バッファは空で、XONでもない）ことを示します。

プログラム例

```
#include        <stdio.h>
#include        <stdlib.h>
#include        <iocslib.h>


main()
{
        void    sub(char *);

        if (OSNS232C() == 0) {
                printf("RS-232C送信ができません¥n");
                exit(1);
        }
        sub("hello¥r¥n");
}


static void     sub(char *str)
{
        while (*str) {
                while (OSNS232C() == 0)
                        ;
                OUT232C(*str++);
        }
}
```

OUT232C

レベル 0

書　式　#include <iocslib. h>

void OUT232C (DATA) ;
int DATA ; /*　送信データ　*/

機　能　RS-232C への送信を行います。

戻り値　戻り値はありません。

プログラム例

```
#include        <stdio.h>
#include        <stdlib.h>
#include        <iocslib.h>


main()
{
        void    sub(char *);

        if (OSNS232C() == 0) {
                printf("RS-232C送信ができません¥n");
                exit(1);
        }
        sub("hello¥r¥n");
}


static void     sub(char *str)
{
        while (*str) {
                while (OSNS232C() == 0)
                        ;
                OUT232C(*str++);
        }
}
```

OUTLPT

レベル0

書　式　#include <iocslib. h>

void OUTLPT (DATA) ;
int DATA ; /*　出力データ　*/

機　能　プリンタへの出力を行います（漢字処理等をしないで直接出力する）。

戻り値　戻り値はありません。

プログラム例

```
#include        <stdio.h>
#include        <stdlib.h>
#include        <iocslib.h>

#define MAXLINE         66      /* 印字可能行数 */
#define MAXWIDTH        132     /* 印字可能文字数 */


main()
{
        void    sub(char *);

        if (INIT_PRN(MAXLINE - 1, MAXWIDTH - 1) == 0) {
                printf("プリンタへの出力はできません¥n");
                exit(1);
        }
        sub("hello¥r¥n");
}


static void     sub(char *str)
{
        while (*str) {
                while (SNSPRN() == 0)
                        ;
                OUTLPT(*str++);
        }
}
```

OUTPRN

レベル 0

書　式　#include <iocslib. h>

void OUTPRN (DATA) ;
int DATA ; /*　出力データ　*/

機　能　プリンタへの出力を行います（シフト JIS で漢字処理あり）。

戻り値　戻り値はありません。

プログラム例

```
#include        <stdio.h>
#include        <stdlib.h>
#include        <iocslib.h>

#define MAXLINE         66      /* 印字可能行数 */
#define MAXWIDTH        132     /* 印字可能文字数 */


main()
{
        void    sub(char *);

        if (INIT_PRN(MAXLINE - 1, MAXWIDTH - 1) == 0) {
                printf("プリンタへの出力はできません¥n");
                exit(1);
        }
        sub("こんにちは¥r¥n");
}


static void     sub(char *str)
{
        while (*str) {
                while (SNSPRN() == 0)
                        ;
                OUTPRN(*str++);
        }
}
```

PAINT

レベル０

書　式

```
#include <iocslib.h>

int PAINT (paintptr) ;
struct PAINTPTR *paintptr;

strct PAINTPTR {
  short x; /*  X座標  */
  short y; /*  Y座標  */
  unsigned short color; /*  パレットコード  */
  unsigned char *buf_start; /*  作業領域先頭アドレス  */
  unsigned char *buf_end; /*  作業領域終了アドレス  */
};
```

機　能　グラフィック画面のペイントを実行します。

戻り値　終了コードを返します。

0：正常終了
−1：グラフィックは使用不可

PATHCHK

レベル 1

書式

```
#include <doslib.h>

int PATHCHK (file, comline, envptr) ;
unsigned char *file ; /* ファイル名格納領域へのポインタ */
struct COMLINE *comline ; /* コマンドライン名格納領域へのポインタ */
unsined char *envptr ; /* 環境文字列へのポインタ */

struct COMLINE {
  unsigned char len ; /* コマンドラインの長さ */
  unsigned char buffer [255] ; /* コマンドライン文字列 */
} ;
```

機能

envptr の環境から path をサーチし、file のコマンド行をファイル名（ドライブ名，パス名つき）とコマンドラインに分けて、それぞれ file と comline が示す領域に格納します。

file は 90 バイト以上、comline は 256 バイト以上必要です。emvptr に 0 を指定すると、呼び出し側プログラムの環境をサーチします。

このファンクションコールを実行後、comline と file を指定して、LOADEXEC、または LOAD を実行した場合、すでに通ったパスであれば、どのパスにおいても簡単に実行することができます。

envptr はグローバル変数 environ が指す C 独自の環境とはちがいます。OS の環境の構造をしたポインタでなければなりません。0 以外を指定する場合は注意してください。

戻り値

負の数ならエラーを表します。

参照関数

LOADEXEC, LOAD, LOADONLY, EXECONLY

プログラム例

```
#include        <stdio.h>
#include        <doslib.h>

main()
{
        struct COMLINE  comline;        /* コマンドライン */
        char    *file[90];                      /* ファイル名 */
        char    *envptr = NULL;         /* 環境文字列 */


        file = "A:¥¥BIN¥¥ED.X A:¥¥CONFIG.SYS";

        if (PATHCHK(file, &comline, envptr) < 0)
                printf("コマンドがみつかりません。¥n")

        printf("file    = %s¥n", file);
        printf("len     = %d¥n", comline.len);
        printf("comline = %s¥n", comline.buffer);
}
```

POINT

レベル0

書式

```
#include <iocslib.h>

int POINT (pointptr) ;
struct POINTPTR *pointptr ;

struct POINTPTR {
  short x ;  /*  X座標  */
  short y ;  /*  Y座標  */
  unsigned short color ;  /*  パレットコード  */
} ;
```

機能 グラフィック画面の指定座標のドットのパレットコードを調べます。

戻り値 終了コードを返します。

0 ： 正常終了
−1 ： グラフィックは使用不可

正常終了した場合には、パレットコードがcolorに格納されます。

PRINT

レベル 1

書　式

```
#include <doslib.h>

void PRINT (msgptr);
unsigned char *msgptr;  /* 表示する文字列へのポインタ */
```

機　能

ヌル文字で終了する文字列を表示します。
このときブレークチェックも行います（^C,^S,^P,^N)。

戻り値

戻り値はありません。

参照関数

GETS

PRNINTST

レベル 0

書　式

```
#include <iocslib.h>

int PRNINTST (ADDRESS) ;
unsigned char *ADDRESS ; /* 割り込み処理アドレス */
```

機　能

プリンタの割り込みを設定します。
ADDRESS には、割り込み処理アドレスを指定します。
0 を指定すると割り込み禁止にします。

戻り値

割り込みが設定されたときは 0、すでに使用中の場合には 0 以外の値を返します。

PRNOUT

レベル 1

書　　式

```
#include <doslib.h>

void PRNOUT (code) ;
int code; /*  出力する文字  */
```

機　　能

プリンタへ1文字出力します。
また、このときブレークチェックも行います (^C)。

戻 り 値

戻り値はありません。

参照関数

PRNSNS, fputc

PRNSNS

レベル 1

書　式　#include 〈doslib. h〉

```
int PRNSNS ( ) ;
```

機　能　プリンタへの出力の可・不可を調べます。

戻り値　出力不可のときには0、出力可能ならば0以外を返します。

PSET

レベル 0

書式

```
#include <iocslib.h>

int PSET (psetptr) ;
struct PSETPTR *psetptr;

struct PSETPTR {
  short x;  /*  X座標  */
  short y;  /*  Y座標  */
  unsigned short color;  /*  パレットコード  */
};
```

機能 グラフィック画面の指定座標に、指定したパレットコードで点を描きます。

戻り値 終了コードを返します。

0 ： 正常終了
−1 ： グラフィックは使用不可

PSPSET

レベル 1

書式

```
#include <doslib.h>

void PSPSET (pspadr) ;
struct PDBADR * pspadr ; /* プロセス管理情報格納領域へのポインタ */
```

機能

pspadrで示すアドレスに、プロセス管理情報を書き込みます。
プロセス管理情報の内容は次の通りです。

```
struct PDBADR  {
  unsigned int env ; /*  環境アドレス  */
  unsigned int exit ; /*  終了アドレス  */
  unsigned int ctrlc ; /*  CTRL―Cアドレス  */
  unsigned int errexit ; /*  エラー中止アドレス  */
  unsigned int comline ; /*  コマンドラインアドレス  */
  unsigned char handle [12] ; /*  ファイルハンドラ使用状況  */
  unsigned int bss ; /*  bss先頭アドレス  */
  unsigned int heep ; /*  heep先頭アドレス ( = bss)  */
  unsigned int stack ; /*  初期stackアドレス ( = heep END)  */
  unsigned int usp ; /*  親プロセスのUSPレジスタ値  */
  unsigned int ssp ; /*  親プロセスのSSP レジスタ値  */
  unsigned short sr ; /*  親プロセスのSRレジスタ値  */
  unsigned short abort_sr ; /* アボート時のSRレジスタ値  */
  unsigned int abort_ssp ; /*  アボート時のSSPレジスタ値  */
  unsigned int trap10 ; /*  POWER OFF or RESET  */
  unsigned int trap11 ; /*  STOPキー (HDOFF処理) 他  */
  unsigned int trap12 ; /*  COPYキー (HCOPY処理) 他  */
  unsigned int trap13 ; /*  CTRL+Cキーでのブレークフラグセット */
  unsigned int trap14 ; /* エラー表示・処理選択(中止・再実行・無視) */
  unsigned int osflg ; /*  起動プロセスフラグ (0 = 子プロセスから、－
                           1 = OSから)  */
  unsigned char reserve [28] ; /*  システムリザーブ  */
  unsigned char exe_path [68] ; /*  ドライブ名およびパス名  */
  unsigned char exe_name [24] ; /*  コマンドファイル名  */
} ;
```

pspadrはメモリ管理関数MALLOCからの戻り値でなければなりません。

戻り値

戻り値はありません。

プログラム例

```
#include        <stdlib.h>
#include        <doslib.h>


main()
{
        struct PDBADR   *psp;   /* プロセス管理情報 */

        psp = (struct PDBADR *)MALLOC(sizeof(struct PDBADR));
        PSPSET(psp);
}
```

PUTCHAR

レベル 1

書　　式	# include <doslib. h> void PUTCHAR (code) ; int code ; /* 表示する文字 */
機　　能	指定された文字コードを表示します。 また、このときブレークチェックも行います (^C,^S,^P,^N)。
戻 り 値	戻り値はありません。
参照関数	GETCHAR, INPOUT, PUTC

PUTGRM

レベル 0

書　　式

```
#include <iocslib.h>

int PUTGRM (putptr) ;
struct PUTPTR * putptr ;

struct PUTPTR {
  short x1 ; /*  始点のX座標  */
  short y1 ; /*  始点のY座標  */
  short x2 ; /*  終点のX座標  */
  short y2 ; /*  終点のY座標  */
  unsigned char  *BUF_START ; /*  バッファ先頭アドレス  */
  unsigned char  *BUF_END ; /*  バッファ終了アドレス  */
} ;
```

機　　能

グラフィック画面の書き出しを行います。
バッファには、データがパックされた型で格納されています。

戻 り 値

終了コードを返します。

　0 ： 正常終了
−1 ： グラフィックは使用不可

RASSIGN

レベル 1

書　　式　# include <doslib. h>

```
int RASSIGN (buff1) ;
unsigned char *buff1; /* 仮想ドライブ名、仮想ディレクトリ名へのポインタ */
```

機　　能　buff1で指定された仮想ドライブ名、仮想ディレクトリ名をキャンセルします。buff1は、MAKEASSIGN関数での1番目のパラメータと同じ内容のパラメータを指定してください。

戻り値　負の数ならエラーを表します。

参照関数　GETASSIGN, MAKEASSIGN

READ

レベル 1

書　式

```
#include <doslib.h>

int READ (fileno, buffer, size) ;
int fileno ;  /*  該当ファイルのファイルハンドル  */
unsigned char *buffer ;  /*  読み込みデータ格納領域へのポインタ  */
int size ;  /*  読み込みバイト数  */
```

機　能

filenoで指定するファイルハンドルから、sizeで指定するバイト数を、bufferが指す領域へ読み込みます。

戻り値

読み込んだバイト数を返します。
これが負の数ならばエラーコードを示します。

参照関数

WRITE

RENAME　レベル 1

書　式

#include <doslib.h>

int RENAME (old, new) ;
unsigned char ＊old ; /＊ 旧ファイル名へのポインタ ＊/
unsigned char ＊new ; /＊ 新ファイル名へのポインタ ＊/

機　能

oldで指定したファイルの名前を、newで示す名前に変更します。
オープンされているファイルに対して実行した場合はエラーになります。

戻り値

正の数（0も）で正常終了、負の数（−1以外も）でエラー。
通常は0が返りますが、特殊デバイスドライバが対象の時は、0以外の正の数の場合があります。

参照関数

MOVE

プログラム例

```
#include        <fcntl.h>
#include        <stat.h>
#include        <io.h>
#include        <doslib.h>


main()
{
        int     exist(char *);
        char    *old = "A:¥¥CONFIG.SYS";
        char    *new = "A:¥¥CONFIG.BAK";


        if (!exist(old))
                RENAME(new, old);       /* 変更(2回め) */
        else
                RENAME(old, new);       /* 変更(1回め) */
}


static int      exist(char *file)
{
        int     fd;

        if ((fd = open(file, O_RDONLY)) < 0)
                return (0);
        else {
                close(fd);
                return (1);
        }
}
```

RMACNV

レベル０

書　式

#include <iocslib. h>

int RMACNV (CODE, W_POINTER, A_POINTER) ;
int CODE ; /*　文字コード　*/
unsigned char *W_POINTER ; /*ワークポインタ　*/
unsigned char *A_POINTER ; /*答えのバッファへのポインタ　*/

機　能

ローマ字変換を行います。
CODE には文字コードを指定します。
W_POINTER にはワークポインタ、A_POINTER には答えのバッファを指すポインタを指定します。

戻り値

変換ステータスを返します。

0 ： 変換の途中
−1 ： 変換不可能
上記以外 ： 変換文字数を示します。(W_POINTER に変換途中の文字が残っている可能性もあります)。

プログラム例

```
#include        <stdio.h>
#include        <iocslib.h>

main()
{
        char    work[10];
        char    ansr[10];
        int     code;           /* 文字コード */

        work[0] = ansr[0] = '¥0';
        code = 'A';

        RMACNV(code, work, ansr);

        printf("%c → %c¥n", code, ansr[0]);
}
```

RMDIR

レベル 1

書　式　#include <doslib. h>

```
int RMDIR (file) ;
unsigned char *file;  /*  ディレクトリ名格納領域へのポインタ  */
```

機　能　ディレクトリを削除します。
ただし、カレントディレクトリや、空でないディレクトリは削除できません。

戻り値　正の数（0も）で正常終了、負の数（−1以外も）でエラー。
通常は0が返りますが、特殊デバイスドライバが対象の時0以外の正の数の場合があります。

参照関数　MKDIR, CHDIR

ROMVER

レベル0

書　　式　#include <iocslib. h>

int ROMVER（）；

機　　能　ROMのバージョンと作成年月日を調べます。

戻 り 値　ROMのバージョンと作成年月日を、下に示すフォーマットで返します。

VVVVVVVV YYYYYYYY MMMMMMMM DDDDDDDD

VVVVVVVV：バージョン（0x10）
YYYYYYYY：年（0x87）
MMMMMMMM：月（0x05）
DDDDDDDD：日（0x07）

プログラム例

```
#include        <stdio.h>
#include        <iocslib.h>

main()
{
        int     verymd;         /* バージョン・年月日 */

        verymd = ROMVER();

        printf("ver.  = %x¥n", (verymd >> 24) & 0xff);
        printf("year  = %x¥n", (verymd >> 16) & 0xff);
        printf("month = %x¥n", (verymd >> 8) & 0xff);
        printf("day   = %x¥n", (verymd >> 0) & 0xff);
}
```

SCROLL

レベル０

書　式

```
#include <iocslib.h>

void SCROLL (MODE, XDOT, YDOT) ;
int MODE; /* モード */
int XDOT; /* Xドット座標 */
int YDOT; /* Yドット座標 */
```

機　能

テキスト/グラフィックの表示座標の設定を行います。
MODEには次の値を設定します。

0：グラフィックスクリーン0設定
1：グラフィックスクリーン1設定
2：グラフィックスクリーン2設定
3：グラフィックスクリーン3設定
8：テキスト設定

戻り値

戻り値はありません。

SEEK

レベル 1

書　式

```
#include <doslib.h>

int SEEK (fileno, offset, mode) ;
int fileno ;  /*  該当ファイルのファイルハンドル  */
int offset ;  /*  ファイルポインタの移動バイト数  */
int mode ;  /*  モード  */
```

機　能

fileno のファイルハンドルで指定されるランダムファイルを、各 mode に従い、offset 分だけファイルポインタを移動します。
モードには次の内容を指定します。

0 ： ファイルの先頭
1 ： ファイルの現在位置
2 ： ファイルの終り

戻り値

現在のファイルポインタの先頭からのオフセットを返します。
負の数ならエラーを表します。

SEND_PR

レベル 1

書　式

```
# include <doslib. h>

int SEND_PR (my_id, your_id, command, buff, length) ;
int my_id ; /* 自分のスレッドの ID */
int your_id ; /* 通信したい相手スレッドの ID */
int command ; /* 通信コマンド */
unsigned char *buff ; /* データバッファへのポインタ */
long length ; /* データバッファのバイト数 */
```

機　能

指定した ID のスレッドにコマンドやデータを送り、SLEEP していたら起こします。

my_id は自分のスレッドの ID です。

your_id は通信したい相手のスレッドの ID です。

command は通信の内容を表すワードの値で、システムで定められているコマンド以外はそれぞれのスレッド間で定義します。

システムで予約してあるコマンドは 0xFF??であり、以下に示す値が定義されています。

0xFFF9 : スレッドを消去するよう要求する

0xFFFB : 強制 SLEEP 状態から起こすのみ（タスク間通信バッファは変化しない）

0xFFFC : SLEEP するよう要求する
起こされたときこのコマンドが送られていたならば、すぐにタスク間通信バッファの your_id を−1 にして SLEEP すべきである
通常、SLEEP しないでタスク間通信バッファを監視している場合に有効

0xFFFF : 処理が終わったかどうか調べるためのコマンド
−28 が返れば、まだ処理中であることを示す

タスク間通信バッファの構造体を以下に示します。

```
struct PRCCTRL {
  long length ;
  unsigned char *buf_ptr ;
  unsigned short command ;
  unsigned short your_id ;
} ;
```

引数のyour_idで指定したスレッドの、タスク間通信バッファのyour_idが−1に設定されていれば、書き込み可能です。
my_idとコマンドをそれぞれ通信バッファのyour_id，commandに設定し、buffからのデータをlengthだけ，通信バッファのbuf_ptrで示されるデータバッファに書き込みます。
そしてlengthを通信バッファのlengthに設定します。
指定したスレッドがSLEEPしていたら起こします。
特に、0xFFFBのコマンドは特殊処理され、指定したスレッドを起こすだけで、タスク間通信バッファのyour_idが−1でなくてもよく、その他のバッファも変化しません。
引数のlengthが、タスク間通信バッファのlengthより大きいときはエラーとなります。
また、書き込み不可の場合もエラーとなります。

戻り値

負の数ならエラーを表します。
正常終了の場合は0を返します。
エラーコードが0xFFFF00??の場合は、指定したidが不正であり、??はidの最大値を示します。
エラーコードが0x80??????の場合は、指定したlengthが不正であり、??????はlengthの最大値を示します。
書き込みエラーの場合は、−28を返します。

参照関数

OPEN_PR, KILL_PR, GET_PR, SUSPEND_PR, SLEEP_PR, TIME_PR, CHANGE_PR

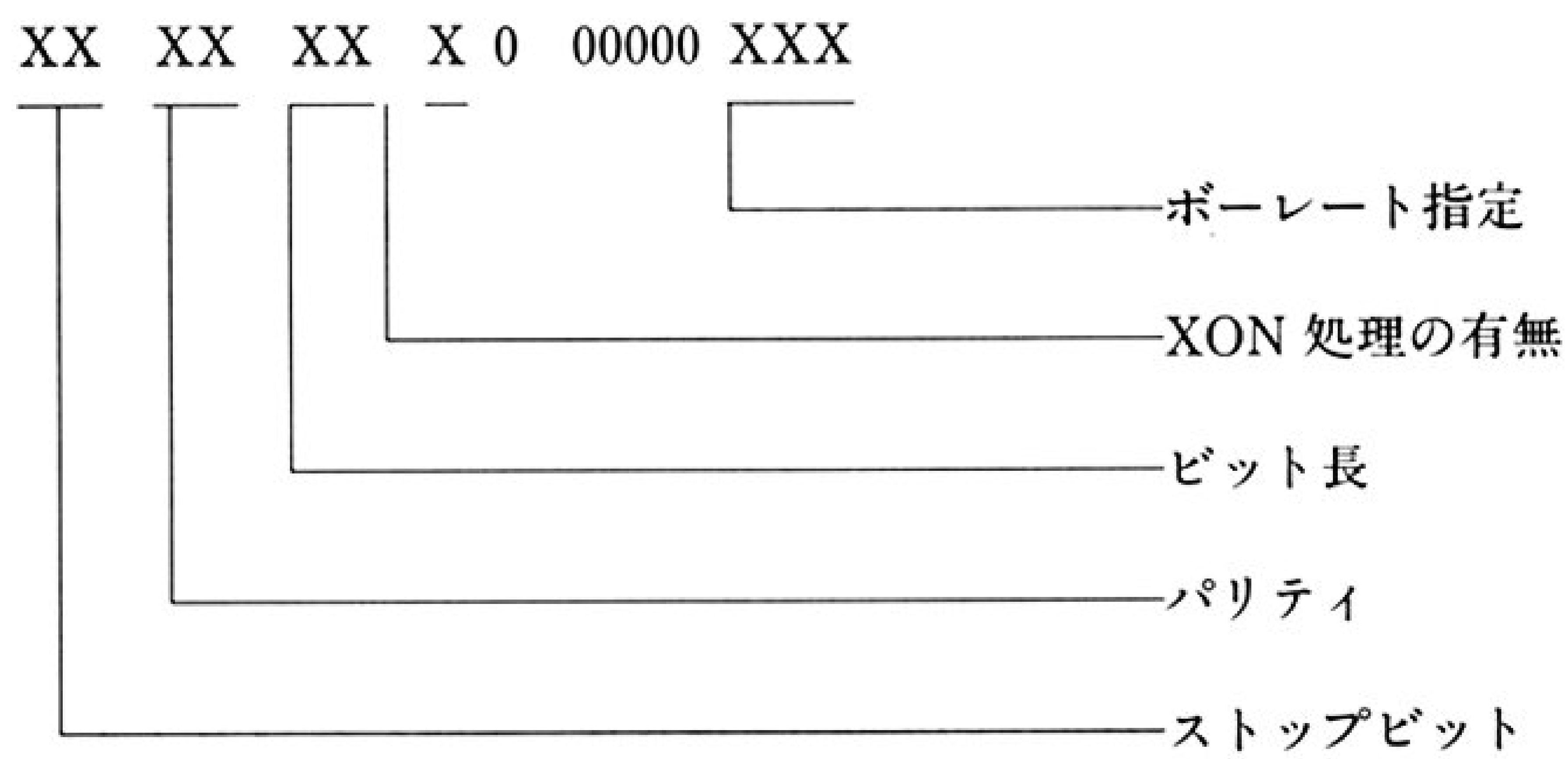
SET232C

レベル 0

書　式　# include 〈iocslib. h〉

int SET232C (MODE) ;
int MODE; /* モード */

機　能　RS-232C のモードを設定します。
MODE の内容と、各項目に設定できる値を次に示します。



●ストップビット
01 ： 1 ビット
10 ： 1.5 ビット
11 ： 2 ビット
00 ： 2 ビット

●パリティ
01 ： 奇数
11 ： 偶数
00 ： パリティなし
10 ： パリティなし

●ビット長
00 ： 5 ビット以下
01 ： 6 ビット
10 ： 7 ビット
11 ： 8 ビット

● XON 処理の有無

0 ： 処理しない

1 ： 処理する

●ボーレート指定

000 ： 75 ボー

001 ： 150 ボー

010 ： 300 ボー

011 ： 600 ボー

100 ： 1200 ボー

101 ： 2400 ボー

110 ： 4800 ボー

111 ： 9600 ボー

戻り値　MODE に−1を指定したときのみ、前のモードを返します。

プログラム例

```
#include        <iocslib.h>

#define STOP1   0x4000  /* ストップビット = 1bit */
#define PEVEN   0x3000  /* パリティ = 偶数 */
#define BITS8   0x0c00  /* ビット長 = 8ビット */
#define XON_E   0x0200  /* xon/off処理 = あり */
#define B9600   0x0007  /* bps = 9600bps */

main()
{
        int     mode;   /* 設定モード */

        mode = STOP1 | PEVEN | BITS8 | XON_E | B9600;
        SET232C(mode);
}
```

SETBLOCK
レベル 1

書　　式

```
#include <doslib.h>

int SETBLOCK (memptr, newlen) ;
int memptr ; /* メモリブロックを指すポインタ */
int newlen ; /* 変更後のメモリサイズ */
```

機　　能

memptr で指定されたメモリブロックのサイズを、newlen に変更します（サイズは、大きくすることも小さくすることもできます）。

memptr は MALLOC が返した値です。

したがって、誤った mempter を指定するとエラーとなります。

newlen に 0x1000000 以上の値を指定したときは、0xFFFFFF として処理します。

戻 り 値

終了コードを返します。

変更できなかったときには 0x81000000＋最大バイト数を返します。

0x8200000?は、まったく変更できないことを示します。

SETDATE

レベル 1

書　式　# include <doslib. h>

int SETDATE (date) ;
int date ; /＊　設定する日付　＊/

機　能　日付を設定します。
dateの形式は次の通りです。

yyyyyyym mmmddddd

ビット 0～4 (ddddd)　：日 (1～31)
ビット 5～8 (mmmm)　：月 (1～12)
ビット 9～15 (yyyyyyy)　：年 (0～99、1980年からの相対年数)

戻り値　正の数 (0も) で正常終了、負の数 (－1以外も) でエラー。

参照関数　GETTIM2, SETTIM2, GETDATE, GETTIME, SETTIME

プログラム例

```
#include        <doslib.h>

main()
{
        int     date;           /* 日付 */
        int     yy = 7;         /* 年 */
        int     mm = 10;        /* 月 */
        int     dd = 5;         /* 日 */

        date = (yy << 9) | (mm << 5) | (dd << 0);

        SETDATE(date);
}
```

SETENV

レベル 1

書　式　# include <doslib. h>

```
int SETENV (name, env, buffer) ;
unsigned char *name; /* 環境変数へのポインタ */
unsigned char *env; /* 環境文字列へのポインタ */
unsigned char *buffer; /* 環境変数格納領域へのポインタ */
```

機　能　evn で指定された環境に変数を設定します。
env に 0 を指定すると、親プロセスの環境に変数を設定します。
buffer に 0 を指定すると、name の環境変数を消去します。
なお、環境変数には 255 バイトまでしか設定できません。

戻り値　負の数ならエラーを表します。

参照関数　GETENV

SETPDB

レベル 1

書　式

```
#include <doslib.h>

int SETPDB (pdbadr) ;
struct PDBADR *pdbadr; /* プロセスのアドレス */
```

機　能

pdbadrで示すプロセスに制御をわたします。
pdbadrは、プログラムの先頭アドレス−0xF0です。
なお、pdbadrはGETPDBが返したポインタでなければなりません。

戻り値

前のpdbadrの値を返します。

参照関数

GETPDB

SETTIM2

レベル 1

書　式

#include <doslib. h>

int SETTIM2 (time) ;
int time ; /＊ 設定する時刻 ＊/

機　能

時刻を time の値に設定します。
time の形式は次の通りです。

00000000 000hhhhh 00mmmmmm : 00ssssss
ビット 0～5 (ssssss) : 秒 (0～59)
ビット 8～13 (mmmmmm) : 分 (0～59)
ビット 16～20 (hhhhh) : 時 (0～23)

戻り値

正の数（0 も）で正常終了、負の数（－1 以外も）でエラー。

参照関数

GETTIM2, GETDATE, SETDATE, GETTIME, SETTIME

プログラム例

```
#include        <doslib.h>

main()
{
        int     time;           /* 設定する時刻 */
        int     hh = 12;        /* 時 */
        int     mm = 50;        /* 分 */
        int     ss = 30;        /* 秒 */

        time = (hh << 16) | (mm << 8) | (ss << 0);
        SETTIM2(time);
}
```

SETTIME

レベル 1

書　式

#include <doslib. h>

int SETTIME (time) ;
int time ; /＊　設定する時刻　＊/

機　能

時刻を設定します。
time の形式は次の通りです。

hhhhhmmm mmmsssss
ビット 0～4 (sssss) ： 秒 (0～29、実際の値は 2 を乗ずる)
ビット 5～10 (mmmmmm) ： 分 (0～59)
ビット 11～15 (hhhhh) ： 時 (0～23)

戻り値

正の数 (0 も) で正常終了、負の数 (－1 以外も) でエラー。

参照関数

GETTIM2, SETTIM2, GETDATE, SETDATE, GETTIME

プログラム例

```
#include        <doslib.h>

main()
{
        int     time;           /* 時刻 */
        int     hh = 10;        /* 時 */
        int     mm = 25;        /* 分 */
        int     ss = 20 / 2;    /* 秒 */

        time = (hh << 11) | (mm << 5) | (ss << 0);
        SETTIME(time);
}
```

SFTJIS

レベル 0

書　式

include <iocslib. h>

int SFTJIS (CODE) ;
int CODE ; /＊　シフト JIS コード　＊/

機　能

シフト JIS コードを JIS コードに変換します。

戻り値

JIS 漢字コードを返します。
ただし、上位 16 ビットが 0xFFFF ならエラーが発生したことを示します。

プログラム例

```
#include        <stdio.h>
#include        <iocslib.h>

main()
{
        int     ms;     /* シフトJISコード */
        int     jis;    /* JIS漢字コード */

        ms = 0x9872;

        jis = SFTJIS(ms);

        printf("0x%04x(sft) : 0x%04x(jis)¥n", ms, jis);
}
```

SKEY_MOD

レベル 0

書　　式

```
#include <iocslib.h>

int SKEY_MOD (MODE, X, Y);
int MODE; /*  モード  */
int X; /*  X座標  */
int Y; /*  Y座標  */
```

機　　能

ソフトキーボードの制御を行います。
MODEには次の内容を指定します。

0：ソフトキーボードの消去
1：ソフトキーボード表示（表示状態では無効）
2：ソフトキーボード表示状態のチェック
−1：ソフトキーボード自動制御（右ボタンで表示/消去）

XにはX座標、YにはY座標を指定します。
なお、MODEが1以外では、X、Yは無意味となります。

戻 り 値

ソフトキーボードの状態を返します。

0：消去状態
1：表示状態

プログラム例

```
#include        <stdio.h>
#include        <iocslib.h>

#define SKEYOFF 0       /* ソフトキーボード消去 */
#define SKEYON  1       /* ソフトキーボード表示 */
#define SKEYCHK 2       /* ソフトキーボード状態チェック */
#define XPOS    12      /* X座標 */
#define YPOS    10      /* Y座標 */


main(int argc, char *argv[])
{
        void    usage(void);
        int     mode;

        if (argc == 1)
                mode = SKEY_MOD(mode, XPOS, YPOS);
        else if (argc == 2) {
                if (strcmp(argv[1], "on") == 0)
                        SKEY_MOD(SKEYON, XPOS, YPOS);
                else if (strcmp(argv[1], "off") == 0)
                        SKEY_MOD(SKEYOFF, XPOS, YPOS);
                else
                        usage();
        }
        else
                usage();
        exit(0);
}


static void     usage()
{
        fprintf(stderr, "Usage: skey_mod [{on|off}]¥n");
        exit(1);
}
```

SKEYSET

レベル０

書　式

```
#include <iocslib.h>

void SKEYSET (KEYCODE) ;
int KEYCODE; /*　キーコード　*/
```

機　能

KEYCODEで指定されたキーを押されたことにします。
KEYCODEには、BITSNS関数で指定するキーコードグループを8倍したものに、そのグループのキーのビット位置の番号を加えたものを指定します。
例えば、'A'ならばグループは3、ビット位置は6なので、3×8+6 = 0x1Eを指定します。

戻り値

戻り値はありません。

プログラム例

```
#include        <iocslib.h>

main()
{
        int     keycode;        /* キーコード */

        keycode = 0x5e;

        SKEYSET(keycode);       /* キーコード入力 */
}
```

SLEEP_PR

レベル 1

書　　式

```
# include <doslib. h>

long SLEEP_PR (time) ;
long time;  /*  待ち時間（単位はミリ秒） */
```

機　　能

SLEEP状態に入ります。
timeには、待ち時間を指定します。
0を指定した場合は永久にSLEEPします。
SLEEP状態に入ったスレッドは、SEND_PR関数を用いて強制的に起こすことができます。
SEND_PR関数のコマンドが0xFFFBの場合、タスク間通信バッファは変化しません。
それ以外のコマンドの場合はバッファに次のように設定されます。

```
struct PRCCTRL  {
  long length;  /*  データバッファに書き込まれたバイト数 */
  unsigned char *buf_ptr;  /*  データバッファの先頭アドレス（変化しない） */
  unsigned short command;  /*  コマンド番号 */
  unsigned short your_id;  /*  起こしてくれたスレッドのID */
}  *buff;
```

SLEEPする前に、データバッファの内容を処理してからデータバッファのアドレスとバイト数を設定し、buff−>your_idに−1を設定してください。
そうすることで、他のアドレスからの通信を許可することになります。
SLEEPしていない時でも、SEND_PR関数によりデータが送られる場合があります。
その場合SLEEPするとすぐ起こされ、設定した待ち時間を戻り値として返します。

戻 り 値

待ち時間が経過して自分で起きた場合は、−1を返します。
タスク間通信バッファの内容は変化しません。
待ち時間が経過したが、SUSPEND_PR関数で止められていたためSEND_PR関数で起こされた場合は、−2を返します。
タスク間通信バッファは、SEND_PR関数により変化します。
戻り値が−1でも−2でもない場合、残り時間をミリ秒単位で返します。
バッファの内容は、SEND_PR関数により変化します。

参照関数　OPEN_PR, KILL_PR, GET_PR, SUSPEND_PR, TIME_PR, CHANGE_PR

S_MALLOC

レベル 1

書　　式

```
# include <doslib. h>

int S_MALLOC (md, size) ;
int md; /*　メモリ割り当てのモード */
int size; /*　確保したい領域のサイズ */
```

機　　能

sizeで指定されたバイト数のメモリ領域を、メインのメモリ管理から確保し、そのポインタを返します。
mdには、以下に示すメモリの割り当て方法を指定します。

0: 下位の方から探し、割り当てる
1: 指定されたサイズを満たすメモリブロックのうち最小のブロックを割り当てる
2: 上位の方から探し、割り当てる

sizeに0x1000000以上の値を指定した時は、0xFFFFFFとして処理します。

S_MALLOCは、通常のアプリケーションプログラムで使用してはいけません。
S_MALLOCを使用するプログラムは、OSから起動されて、終了することができないプログラムでなければなりません。

戻 り 値

メモリが確保できた時は、確保できた領域を指すポインタを返します。
確保できなかった時は、0x81000000＋最大バイト数を返します。
0x8200000?は全く確保できないことを表します。

参照関数

MALLOC, MFREE, S_MFREE, S_PROCESS

S_MFREE　レベル 1

書　式

```
# include <doslib.h>

int S_MFREE (memptr) ;
int memptr ;  /* メモリブロックを指すポインタ */
```

機　能

メインのメモリ管理のポインタをmemptrで指定して、メモリブロックを開放します。
memptrはS_MALLOC関数が返した値でなければならないので、誤ったmemptrを指定するとエラーになります。
memptrがS_PROCESS関数で指定しているアドレスで、かつそのスレッドIDがカレントIDならば、自分自身を削除します（KILL_PR関数）。
その時、内部のメモリ管理で常駐終了しているプロセスなどは、メインのメモリ管理に連結されます。
この場合、KILL_PRと同じ動作になりますので、注意してください。

また、S_MFREEを行う場合は、指定したmemptrのメモリブロック中で動作しているプログラムが存在していない状態で行うようにしてください。

戻り値

正の数（0も）で正常終了、負の数（−1以外も）でエラー。

参照関数

MALLOC, MFREE, S_MALLOC, S_PROCESS

SNSPRN

レベル 0

書　　式　# include <iocslib. h>

int SNSPRN ( ) ;

機　　能　プリンタ出力が可能か否かをチェックします。

戻 り 値　0 以外ならばプリンタへの出力は可能です。
0 のときには出力できません。

プログラム例

```
#include        <stdio.h>
#include        <stdlib.h>
#include        <iocslib.h>

#define MAXLINE         66      /* 印字可能行数 */
#define MAXWIDTH        132     /* 印字可能文字数 */


main()
{
        void    sub(char *);

        if (INIT_PRN(MAXLINE - 1, MAXWIDTH - 1) == 0) {
                printf("プリンタへの出力はできません¥n");
                exit(1);
        }
        sub("こんにちは¥r¥n");
}


static void     sub(char *str)
{
        while (*str) {
                while (SNSPRN() == 0)
                        ;
                OUTPRN(*str++);
        }
}
```

SPALET　レベル 0

書　式　#include <iocslib. h>

```
int SPALET (CODE, BLOCK, COLORCODE) ;
int CODE ; /*  垂直帰線期間との同期  */
int BLOCK ; /*  パレットブロック  */
int COLORCODE ; /*  カラーコード  */
```

機　能　スプライトパレットの設定と読み出しを行います。
CODE には、垂直帰線期間と同期してパレットを変えるか、同期を取らずに変えるかを指定します。
なお、垂直帰線期間を検出せずにパレットコードを変えると、画面にちらつきが生じます。

0x00～0x0F　垂直帰線期間を検出後に設定
0x80～0x8F　検出なし

BLOCK にはパレットブロックを指定します（1～15）。
COLORCODE にはカラーコードを指定します（0～65535）。
−1 を指定した場合には、カラーコードは変更せずにカラーコードの読み出しのみ行います。

戻り値　設定前のカラーコード、またはエラーコード（−2 ＝ パラメータエラー、パレットブロック 0 をアクセスした）を返します。

SP_CGCLR

レベル 0

書　式

```
#include <iocslib.h>

int SP_CGCLR (CODE) ;
int CODE; /* パターンコード */
```

機　能

PCG をクリアします。
CODE にはパターンコードを指定します。(0～255、16×16 ドットパターン)。

戻り値

終了コードを返します。

0 ： 正常終了
−1 ： 画面モードエラー (画面サイズが 768×512 のときなど)

SP_DEFCG

レベル 0

書式 #include <iocslib. h>

```
int SP_DEFCG (CODE, SIZE, ADDRESS) ;
int CODE; /* パターンコード */
int SIZE; /* パターンサイズ */
unsigned char *ADDRESS; /* パターンデータバッファのアドレス */
```

機能 PCGの設定を行います。

CODEにはパターンコードを指定します（0〜255)。

SIZEにはパターンサイズを指定します。

0 ： 8×8ドットパターン

1 ： 16×16ドットパターン

ADDRESSにはパターンデータバッファのアドレスを指定します。

このときパターンデータバッファのSIZEは、は次のようになります。

0 ： 32バイト

1 ： 128バイト

戻り値 終了コードを返します。

0 ： 正常終了

−1 ： 画面モードエラー（画面サイズが768×512のときなど）

SP_GTPCG

レベル0

書　　式

```
#include <iocslib.h>

int SP_GTPCG (CODE, SIZE, ADDRESS) ;
int CODE;  /*  パターンコード  */
int SIZE;  /*  パターンサイズ  */
unsigned char *ADDRESS;  /*  パターンデータバッファのアドレス  */
```

機　　能

PCGの読み出しを行います。
CODEにはパターンコードを指定します（0～255)。
SIZEにはパターンサイズを指定します。

0：8×8ドットパターン
1：16×16ドットパターン

ADDRESSには、パターンデータバッファのアドレスを指定します。
このときパターンデータバッファのSIZEは、次のようになります。

0：32バイト
1：128バイト

戻 り 値

終了コードを返します。

0：正常終了
−1：画面モードエラー（画面サイズが768×512のときなど）

正常終了の場合、パターンデータバッファにデータが格納されます。

SP_INIT

レベル 0

書　式　# include <iocslib. h>

```
int SP_INIT ( ) ;
```

機　能　スプライト面の初期化を行います。

戻り値　終了コードを返します。

0 ： 正常終了
−1 ： 画面モードエラー　(画面サイズが 768×512 のときなど)

SP_OFF

レベル 0

書　式

```
#include <iocslib.h>

void SP_OFF ( ) ;
```

機　能　スプライト面の表示を消します。

戻り値　戻り値はありません。

SP_ON

レベル 0

書　　式　#include <iocslib.h>

```
int SP_ON ( ) ;
```

機　　能　スプライト面の表示を行います。

戻 り 値　終了コードを返します。

0 ： 正常終了
−1 ： 画面モードエラー（画面サイズが 768×512 のときなど）

SP_REGGT

レベル0

書　式

```
#include <iocslib.h>

int SP_REGGT (SPR_NO, X, Y, CODE, PRW);

int SPR_NO;  /* スプライトプレーン番号 */
int *X;  /* X座標 */
int *Y;  /* Y座標 */
int *CODE;  /* パターンコード */
int *PRW;  /* プライオリティ */
```

機　能

スプライトレジスタの読み出しを行います。
SPR_NOにはスプライトプレーン番号を指定します（0～127)。
XにはX座標、YにはY座標を返すポインタを指定します（0～1023)。
PRWにはプライオリティを返すポインタを指定します（0～3)。
CODEには次の内容を返すポインタを指定します。

ビット16～32：常に0
ビット15：垂直反転（0＝標準、1＝反転）
ビット14：水平反転（0＝標準、1＝反転）
ビット12～13：常に0
ビット8～11：パレットブロック番号（0～15)
ビット0～7：パターンコード（0～255)

戻り値

終了コードを返します。

0：　正常終了
−1：　画面モードエラー（画面サイズが768×512のときなど）

正常終了の場合、X座標、Y座標、パターンコード、プライオリティがそれぞれに格納されます。

SP_REGST

レベル０

書　式

```
#include <iocslib.h>

int SP_REGST (SPRITENO, MODE, X, Y, CODE, PRW) ;
int SPRITENO ;  /*  スプライトプレーン番号  */
int MODE ;  /*  垂直帰線期間の検出モード  */
int X ;  /*  X座標  */
int Y ;  /*  Y座標  */
int CODE ;  /*  パターンコード  */
int PRW ;  /*  プライオリティ  */
```

機　能

スプライトレジスタの設定を行います。
SPRITENOには、スプライトプレーン番号を指定します（0～127）。
MODEには、垂直帰線期間の検出の有無を指定します（最上位ビットが0ならば検出後に設定、1ならば検出なし）。
XにはX座標、YにはY座標をそれぞれ0～1023の範囲内で指定します。
－1が指定された場合には、設定を変更せずに現在値を使用します。
CODEには、パターンコードを指定します。
－1が指定された場合には、設定を変更せずに現在値を使用します。
詳しくは、SP_REGGT関数を参照してください。
PRWには、プライオリティを指定します（0～3）。
－1が指定された場合には、設定を変更せずに現在値を使用します。

戻り値

終了コードを返します。

0：正常終了
－1：画面モードエラー（画面サイズが768×512のときなど）

S_PROCESS

レベル 1

書　式

```
# include <doslib. h>

int S_PROCESS (id, start, size, i_len) ;
int id; /*  スレッド ID */
int start; /*  サブのメモリ管理の先頭アドレス */
int size; /*  サブのメモリ管理のバイト数 */
int i_len; /*  先頭ブロックのバイト数 */
```

機　能

サブのメモリ管理領域を設定します。
id にはスレッド ID を指定します。
start にはサブのメモリ管理の先頭アドレス、size にはその領域のバイト数を指定します。
指定された ID のスレッドでのメモリ管理は、以後この範囲内となります。
i_len には、新しいメモリ管理の先頭のメモリブロック（必ず確保される）のバイト数を指定します。
また、start からの最初の 16 バイトは、メモリ管理領域として使用されるので破壊されます。
なお、id = 0（メインスレッド）は、S_PROCESS 関数で変更することはできません。

start には、S_MALLOC で返されたメモリブロックのアドレスを指定します。
S_PROCESS は、通常のアプリケーションプログラムで使用してはいけません。
S_PROCESS を使用するプログラムは、OS から起動されて、終了することができないプログラムでなければなりません。

戻り値

正常終了なら i_len で指定したバイト数のメモリブロックの先頭を指すポインタを返します。
戻り値が 0xFFFF00?? なら ID エラーです。
戻り値が −14 なら、size が i_len+16 より小さいことを示します。

参照関数

MALLOC, MFREE, S_MALLOC, S_MFREE

SUPER

レベル 1

書式

include <doslib. h>

int SUPER (stack) ;
int stack ; /＊ 切り替えモード ＊/

機能

スーパーバイザモード/ユーザーモードの切り替えを行います。
stack が 0 のときは、USP の値を SSP にセットして、スーパーバイザモードに切り替えます。
stack が 0 以外のときは、stack を SSP にセットして、ユーザーモードに切り替えます。

戻り値

stack が 0 のときは前の SSP の値を返します。
負の数ならエラーを表します。

プログラム例

```
#include        <doslib.h>

main()
{
        /* ユーザモード → スーパーバイザモード */
        SUPER(0);
}
```

SUPER_JSR

レベル 1

書　式

```
# include <doslib. h>

void SUPER_JSR (func, inregs, outregs) ;
void  (*func) ( ) ;  /*  プログラム先頭アドレス  */
struct DREGS  {
  int d0;
  int d1;
  int d2;
  int d3;
  int d4;
  int d5;
  int d6;
  int d7;
  int a0;
  int a1;
  int a2;
  int a3;
  int a4;
  int a5;
  int a6;
}    *inregs;  /*  プログラムに渡すレジスタの値  */
struct DREGS  *outregs;  /*  プログラムが返すレジスタの値  */
```

機　能

スーパーバイザ領域のプログラムをサブルーチンコールします。
funcは、呼び出したいプログラムのエントリアドレスです。
inregsには、呼び出しプログラムに渡す各レジスタの初期値の格納領域を指すポインタを指定します。
outregsには、呼び出したプログラムから制御が戻るときのレジスタの状態を格納する、バッファの先頭のポインタを指定します。
なお、srは変化しません。
usp/sspはどのような値が渡されるか不定なので、スタックを使って引数を渡すことはできません。
なお、サブルーチンコールの結果、暴走やバスエラーが発生しても処理していません。

戻り値

戻り値はありません。

SUSPEND_PR

レベル 1

書　式

```
# include <doslib. h>

int SUSPEND_PR (id) ;
int id;  /*  スレッドの ID  */
```

機　能

指定した ID のスレッドを強制的に SLEEP 状態にします。
SLEEP 状態になったスレッドは SEND_PR で起こされるまで SLEEP します。

戻り値

正常終了なら 0 を返します。
負の数ならエラーを表します。
ID がおかしい場合は 0xFFFF00??を返します。??が ID の最大値です。
自分自身を指定した場合は−1 を返します。

参照関数

OPEN_PR, KILL_PR, GES_PR, SLEEP_PR, SEND_PR, TIME_PR, CHANGE_PR

SYMBOL

レベル0

書　式

```
#include <iocslib.h>

int SYMBOL (symbolptr);
struct SYMBOLPTR {
  short x1;  /*  X座標  */
  short y1;  /*  Y座標  */
  unsigned char *string_address;  /*  文字列アドレス  */
  unsigned char mag_x;  /*  横倍率  */
  unsigned char mag_y;  /*  縦倍率  */
  unsigned short color;  /*  パレットコード  */
  unsigned char font_type;  /*  文字フォントのタイプ  */
  unsigned char angle;  /*  角度コード  */
} *symbolptr;
```

機　能

グラフィック画面のシンボルを実行します。

X1、Y1には、グラフィック座標を指定します。

座標は、通常は表示する文字列の左上隅の座標を指定しますが、回転角度が90度、180度、270度のときはそれぞれ、左下、右下、右上隅の座標を指定しなければなりません。

mag_x、mag_yには、文字の拡大倍率を指定します。

mag_xは横の倍率、mag_yは縦の倍率ですが、これは文字の縦横に対しての指定で、画面の縦横に対してではないことに注意してください。

colorには、文字の色を指定します。

font_typeには次の文字フォントのタイプを指定します。

0：12×12ドットフォント
1：16×16ドットフォント
2：24×24ドットフォント

angleには次の角度コードを指定します。

0：通常
1：90度回転
2：180度回転
3：270度回転

戻り値

終了コードを返します。

0：正常終了
−1：グラフィックは使用不可

TCOLOR

レベル 0

書式 #include <iocslib. h>

```
void TCOLOR (TEXTCOLOR) ;
int TEXTCOLOR; /* テキストのカラー */
```

機能 テキストのカラーを指定します。

TEXTGET, TEXTPUT, CLIPPUT などの関数が使用するテキスト画面を決めます。

TEXTCOLOR には 0～15 まで指定できますが、原則として 1、2、4、8のみ指定してください。

1 ： 0xE00000～0xE1FFFF の面を使用
2 ： 0xE20000～0xE3FFFF の面を使用
4 ： 0xE40000～0xE5FFFF の面を使用
8 ： 0xE60000～0xE7FFFF の面を使用

戻り値 戻り値はありません。

TEXTGET

レベル 0

書式

```
#include <iocslib.h>

void TEXTGET (XDOT, YDOT, BUF) ;
int XDOT ; /*  Xドット座標  */
int YDOT ; /*  Yドット座標  */
struct FNTBUF *BUF ; /*  データバッファアドレス  */
struct FNTBUF {
  short xl ; /*  X方向のドット数  */
  short yl ; /*  Y方向のドット数  */
  unsigned char buffer [72] ; /*  データバッファ  */
} ;
```

機能

XDOT および YDOT で指定されたドット座標から、BUF で指定されたアドレスへパターンを読み込みます。

BUF にはスーパーバイザ領域のアドレスも指定できますが、誤ったアドレスを指定しないように注意してください。

BUF には、パターンの X 方向のドット数、Y 方向のドット数をそれぞれ short で指定してこの関数を呼びます。

buffer は読み込んだパターンが入るだけの充分な大きさが必要です。

構造体 FNTBUF のデータサイズを越える場合には、メモリ確保用の関数 malloc などで確保した領域を使用してください。

戻り値

戻り値はありません。

TEXTPUT

レベル 0

書式 #include <iocslib. h>

```
void TEXTPUT (XDOT, YDOT, BUF) ;
int XDOT ; /*  X ドット座標  */
int YDOT ; /*  Y ドット座標  */
struct FNTBUF *BUF ; /*  データバッファアドレス  */
struct FNTBUF {
  short xl ; /*  X 方向のドット数  */
  short yl ; /*  Y 方向のドット数  */
  unsigned char buffer [72] ; /*  データバッファ  */
} ;
```

機能 BUF で指定されたアドレスから、XDOT および YDOT で指定されたドット座標へ、パターンを書き出します。

BUF にはスーパーバイザ領域のアドレスも指定できますが、誤ったアドレスを指定しないよう注意してください。

BUF には、パターンの X 方向のドット数、Y 方向のドット数、およびパターンデータを格納してください。

buffer には、指定した X、Y 方向のドット数に必要なだけのデータを格納しておいてください。

構造体 FNTBUF のデータサイズを越える場合には、メモリ確保用の関数 malloc などで確保した領域を使用してください。

戻り値 戻り値はありません。

TGUSEMD

レベル 0

書　式

```
#include <iocslib.h>

int TGUSEMD (TEXT_GR, MODE) ;
int TEXT_GR ; /*　画面の種類　*/
int MODE ; /*　画面のモード　*/
```

機　能

TEXT_GRで指定した画面のモードの設定、または読みとりを行います。
TEXT_GR、MODEに指定できる内容は次の通りです。

● TEXT_GR
- 0：グラフィック (0xC00000～0xDFFFFF)
- 1：テキスト (0xE40000～0xE7FFFF)

● MODE
- 0：誰も使用していない
- 1：システムで使用している
  (ソフトキー/電卓/RAMディスク)
- 2：アプリケーションで使用している
- 3：アプリケーションで使用して、壊れたままである
- −1：画面のモードを読みとる

ただしTEXT_GRに1を指定した場合、MODEに0または3を指定するとMODEに1を指定したことになります。

戻り値

画面のモードを返します。

プログラム例

```
#include        <stdio.h>
#include        <iocslib.h>

#define M_GRPH  0       /* グラフィックモード */
#define M_TEXT  1       /* テキストモード */
#define S_NOT   1       /* 誰も使用していない */
#define S_SYS   2       /* システムが使用 */
#define S_APP   3       /* アプリケーションが使用 */
#define S_APPD  4       /* 壊れたまま */
#define S_READ  (-1)    /* 画面モード読み取り */

main()
{
        int     r;

        if ((r = TGUSEMD(M_TEXT, S_READ)) == S_NOT)
                puts("使用されていません。");
        else if (r == S_SYS)
                puts("システムで使用中。");
        else
                puts("アプリケーションで使用中。");
}
```

TIMEASC

レベル0

書　式　#include <iocslib.h>

int TIMEASC (BINTIME, BUF) ;
int BINTIME ;　/*　時刻 (2進数)　*/
unsigned char *BUF ;　/*　文字列格納領域へのポインタ　*/

機　能　2進データを、時刻を表す文字列に変換し、格納します。
BINTIMEの形式と指定内容は次の通りです。

00000000 hhhhhhhh mmmmmmmm ssssssss

hhhhhhhh ： 時 (00～23)
mmmmmmmm ： 分 (00～59)
ssssssss ： 秒 (00～59)

BUFには、時刻を表す文字列の格納領域の先頭アドレスを指定します。
この領域は、最低9バイト以上確保してください。
BUFにはスーパーバイザ領域のアドレスも指定できますが、誤ったアドレスを指定しないよう注意してください。

BUFには、次のような形式で格納されます。

"12:05:30"

戻り値　エラーが発生したときは−1を返します。

プログラム例

```
#include        <stdio.h>
#include        <iocslib.h>

main()
{
        char    str[9]; /* 文字列格納領域 */
        int     bin;    /* 時刻(2進数) */

        bin = TIMEBIN(TIMEGET());

        TIMEASC(bin, str);

        printf("%s¥n", str);
}
```

TIMEBCD

レベル 0

書　式　#include <iocslib. h>

int TIMEBCD (BINTIME) ;
int BINTIME; /* 時刻 (2 進数) */

機　能　2 進数の時刻を、時計に設定できる形式に変換します。
BINTIME の形式と指定内容は次の通りです。

00000000 hhhhhhhh mmmmmmmm ssssssss

hhhhhhhh : 時 (00～23)
mmmmmmmm : 分 (00～59)
ssssssss : 秒 (00～59)

上記の時・分・秒はすべて 2 進数です。

戻り値　次の内容を返します。

0000TTTT HHHHHHHH MMMMMMMM SSSSSSSS

TTTT : 1 (24 時間計であることを示します)
HHHHHHHH : 時 (00～23)
MMMMMMMM : 分 (00～59)
SSSSSSSS : 秒 (00～59)

上記の時，分，秒はすべて BCD2 桁です。
エラーが発生したときは－1 を返します。

プログラム例

```
#include        <iocslib.h>

#define HOU     10
#define MIN     28
#define SEC     40

main()
{
        int     bin;    /* 時刻(2進数) */
        int     bcd;    /* 時刻(BCD) */

        bin = (HOU << 16) | (MIN << 8) | (SEC << 0);

        bcd = TIMEBCD(bin);

        TIMESET(bcd);   /* 時刻を時計に設定 */
}
```

TIMEBIN

レベル 0

書　式

\# include <iocslib. h>

int TIMEBIN (BCDTIME) ;
int BCDTIME ; /＊ 時刻 (BCD) ＊/

機　能

時刻を BCD から 2 進数に変換します。
BCDTIME の形式と指定内容は次の通りです。

00000000 HHHHHHHH MMMMMMMM SSSSSSSS

HHHHHHHH : 時 (00～23)
MMMMMMMM : 分 (00～59)
SSSSSSSS : 秒 (00～59)

上記の時・分・秒はすべて BCD2 桁です。

戻り値

次の内容を返します。

00000000 hhhhhhhh mmmmmmmm ssssssss

hhhhhhhh : 時 (00～23)
mmmmmmmm : 分 (00～59)
ssssssss : 秒 (00～59)

上記の時，分，秒はすべて 2 進数です。

プログラム例

```
#include        <stdio.h>
#include        <iocslib.h>

main()
{
        int     bcdtime;            /* 時刻(BCD) */
        int     bintime;            /* 時刻(2進数) */

        bcdtime = TIMEGET();

        bintime = TIMEBIN(bcdtime);
        printf("%d時", (bintime >> 16) & 0xff);
        printf("%d分", (bintime >> 8) & 0xff);
        printf("%d秒", (bintime >> 0) & 0xff);
        printf("¥n");
}
```

TIMECNV

レベル0

書　式

include <iocslib. h>

int TIMECNV (ADDRESS) ;
unsigned char ＊ADDRESS ; /＊ 時刻を示す文字列へのポインタ ＊/

機　能

時刻を表す文字列を2進データに変換します。
ADDRESSには、時刻を表す文字列の先頭アドレスを指定します。
ADDRESSにはスーパーバイザ領域のアドレスも指定できますが、誤ったアドレスを指定しないよう注意してください。
また、時刻を表す文字列は次のように指定します。

"12 : 05 : 30"

区切り記号は：(コロン)以外でもかまいません。

戻り値

2進数に変換した時刻を返します。
時刻の形式は次の通りです。

00000000 hhhhhhhh mmmmmmmm ssssssss

hhhhhhhh ： 時 (00～23)
mmmmmmmm ： 分 (00～59)
ssssssss ： 秒 (00～59)

エラーが発生したときは−1を返します。

プログラム例

```
#include        <iocslib.h>

main()
{
        int     bcd;    /* 時刻(BCD) */
        char    *str;

        str = "10:08:40";

        bcd = TIMEBCD(TIMECNV(str));

        TIMESET(bcd);
}
```

TIMEGET

レベル 0

書　式　# include <iocslib. h>

int TIMEGET ( ) ;

機　能　時計から時刻を読み込みます。

戻り値　次の内容を返します。

00000000 HHHHHHHH MMMMMMMM SSSSSSSS

HHHHHHHH ： 時 (00～23)
MMMMMMMM ： 分 (00～59)
SSSSSSSS ： 秒 (00～59)

上記の時，分，秒はすべて BCD2 桁です。

プログラム例

```
#include        <stdio.h>
#include        <iocslib.h>

main()
{
        int     bcd;    /* 時刻(2進数) */
        int     bin;    /* 時刻(BCD) */

        bcd = TIMEGET();

        bin = TIMEBIN(bcd);

        printf("%d時", (bin >> 16) & 0xff);
        printf("%d分", (bin >> 8) & 0xff);
        printf("%d秒", (bin >> 0) & 0xff);
        printf("¥n");
}
```

TIME_PR

レベル 1

書　式　# include <doslib. h>

long TIME_PR ( ) ;

機　能　現在のタイマーのカウンタ値（ミリ秒単位）を返します。
バックグラウンドで複数のスレッドが並行して動作している場合、プログラムで一定の時間を計るためには、どのスレッドが動作していても、一定して変化するカウンタが必要になります。
longでの最大値を越えると0に戻るので、そのまま前回に返されたタイマーのカウンタ値との引き算で経過時間が分かります。

戻り値　現在のタイマーのカウンタ値を返します。

参照関数　OPEN_PR, KILL_PR, GET_PR, SUSPEND_PR, SLEEP_PR, SEND_PR, CHANGE_PR

TIMERDST

レベル 0

書　式

```
#include <iocslib.h>

int TIMERDST (ADDRESS, MODE, COUNTER);
unsigned char *ADDRESS;  /* 割り込み処理アドレス */
int MODE;  /* モード */
int COUNTER;  /* カウンタ */
```

機　能

MFP の TIMER-D による割り込みの設定を行います。
ADDRESS には割り込み処理アドレスを指定します（0 ならば割り込み禁止）。
MODE には次の内容を指定します。

1 ： 1μsec 単位
2 ： 2.5μsec 単位
3 ： 4μsec 単位
4 ： 12.5μsec 単位
5 ： 16μsec 単位
6 ： 25μsec 単位
7 ： 50μsec 単位

COUNTER には 0 から 255 までの値を指定します。
MODE の値に COUNTER の値を乗じた値の単位で、割り込みが行われます。
なお、Human68k ver. 2.0 の動作中は、割り込みを変更できません。

戻り値

割り込みが可能なときは 0、すでに使用中の場合には 0 以外の値を返します。

TIMESET

レベル0

書　式　#include <iocslib. h>

void TIMESET (BCDTIME) ;
int BCDTIME ; /* 時刻 (BCD) */

機　能　時計に時刻を設定します。
BCDTIMEの形式と指定内容は次の通りです。

0000TTTT HHHHHHHH MMMMMMMM SSSSSSSS

TTTT	:	1:24時間計
HHHHHHHH	:	時 (00～23)
MMMMMMMM	:	分 (00～59)
SSSSSSSS	:	秒 (00～59)

上記の時・分・秒はすべてBCD2桁です。

戻り値　戻り値はありせん。

プログラム例

```
#include        <iocslib.h>

#define HOU             10
#define MIN             8
#define SEC             40
#define CLOCK24         0x1000000
#define CLOCK12         0x0000000

main()
{
        int     bin;    /* 時刻(2進数) */
        int     bcd;    /* 時刻(BCD) */

        bin = CLOCK24 | (HOU << 16) | (MIN << 8) | (SEC << 0);

        bcd = TIMEBCD(bin);

        TIMESET(bcd);   /* 時刻を時計に設定 */
}
```

TPALET

レベル 0

書式

```
#include <iocslib. h>

int TPALET (PALETNO, COLORCODE) ;
int PALETNO ;  /*  パレット番号   */
int COLORCODE ; /*  カラーコード   */
```

機能

テキストのパレットを設定します。

PALETNO はテキストパレットの番号です。0〜15 を指定します。

テキストパレットの 0〜3 は、テキストに表示する文字の色指定に使われます。

テキストパレットが 4〜7、8〜15 の場合にはそれぞれ 4〜7、8〜15 に同一のカラーコードが設定されます。

4〜7、8〜15 は、ソフトキーボード/電卓で使用されます。

0〜3 ： テキストカラー 0〜3

4〜7 ： ソフトキーボード/電卓カラー
(4〜7 に同じカラーコードをセット)

8〜15 ： ソフトキーボード/ 電卓カラー
(8〜15 に同じカラーコードをセット)

COLORCODE には 0〜65535 を指定します。

なお、COLORCODE に −1 を設定するとパレットの読み出しのみ行い、−2 ならばパレットをシステムの既定値にします。

戻り値

COLORCODE が −1 のときには、現在のカラーコードを返します。

TPALET2

レベル 0

書　　式　#include <iocslib. h>

int TPALET2 (PALETNO, COLORCODE) ;
int PALETNO ; /＊　パレット番号　＊/
int COLORCODE ; /＊　カラーコード　＊/

機　　能　テキストのパレットを設定します。
PALETNO には、0～15 を指定します。
COLORCODE には、0～65535 を指定します。
なお、COLORCODE に－1 を設定するとパレットの読み出しのみ行い、－2 ならばパレットをシステムの既定値に戻します。

戻 り 値　COLORCODE が－1 のときは、現在のカラーコードを返します。

プログラム例

```
#include        <stdio.h>
#include        <iocslib.h>


main()
{
        int     pno;    /* パレット番号 */
        int     color;  /* カラーコード */

        for (pno = 0; pno < 16; pno++) {
                /* パレット読出し */
                color = TPALET2(pno, -1);
                printf("palet No.%d の ", pno);
                printf("ColorCode: 0x%x¥n", color);
        }
}
```

TRAP15

レベル 0

書　式

```
# include <iocslib. h>

int TRAP15 (inregs, outregs) ;
struct REGS  {  /*  レジスタを直接アクセスするための構造体  */
  int d0 ;
  int d1 ;
  int d2;
  int d3 ;
  int d4 ;
  int d5 ;
  int d6 ;
  int d7 ;
  int a1 ;
  int a2 ;
  int a3 ;
  int a4 ;
  int a5 ;
  int a6 ;
}  *inregs ;  /*  呼び出しプログラムに渡すレジスタの値を格納した構造体
                  へのポインタ  */
struct REGS  *outregs ;  /*  呼び出しプログラムから返されるレジスタの値
                             が格納される構造体へのポインタ  */
```

機　能

inregsで指定した構造体中のデータを各レジスタに書き込んだ後、直接trap # 15 命令を実行します。

そして、実行後のレジスタの値を outregs で指定された構造体へ格納します。

引数であるレジスタの初期値は自分で確保した構造体に格納し、さらに TRAP15 にはその構造体のポインタを渡します。

trap # 15 命令は通常 IOCS コールを実行するので、inregs－＞d0 に IOCS コール番号を書き込み、他のメンバに必要な値を書き込んだ後で、TRAP15 関数を実行することで直接 IOCS コールを行うことができます。

inregs に格納した値がおかしいと暴走する可能性があるので十分注意して使用してください。

戻り値

trap # 15 命令により呼び出された IOCS コールが返す d0.l の値を返します。

プログラム例

```
#include        <stdio.h>
#include        <iocslib.h>


main()
{
        struct REGS     inregs;
        struct REGS     outregs;
        int     r;

        inregs.d0 = 0x01;
        r = TRAP15(&inregs, &outregs);
        if (r == 0)
                printf("キーが押されていない¥n");
        else {
                printf("スキャンコード0x%x¥n", (r >> 8) & 0xff);
                printf("内部コード    0x%x¥n", r & 0xff);
        }
}
```

TVCTRL

レベル 0

書式

```
#include <iocslib.h>

void TVCTRL (CODE) ;
int CODE; /*  TV コントロールコード  */
```

機能

CODEにTVコントロールコード(6 ビット)を指定し、TVの操作を行います。TV コントロールコードは次の通りです。

0x01	:	ボリュームを上げる
0x02	:	ボリュームを下げる
0x03	:	ボリュームを標準にする
0x04	:	チャンネルコール
0x05	:	テレビ画面 (初期化・リセット)
0x06	:	音声ミュート
0x07	:	電源 ON
0x08	:	テレビ/コンピュータ
0x09	:	テレビ/外部、コンピュータノーマル/オーバー
0x0A	:	コントラスト標準
0x0B	:	チャンネルアップ
0x0C	:	チャンネルダウン
0x0D	:	電源 OFF
0x0E	:	電源 ON/OFF
0x0F	:	スーパー 1
0x10	:	チャンネル 1
0x11	:	チャンネル 2
0x12	:	チャンネル 3
0x13	:	チャンネル 4
0x14	:	チャンネル 5
0x15	:	チャンネル 6
0x16	:	チャンネル 7
0x17	:	チャンネル 8
0x18	:	チャンネル 9
0x19	:	チャンネル 10
0x1A	:	チャンネル 11
0x1B	:	チャンネル 12
0x1C	:	テレビ画面 (0x05)
0x1D	:	コンピュータ画面 (0x05+0x08)

0x1E ： スーパー1 (0x05+0x0F)
0x1F ： スーパー2 (0x05+0x0F+0x0A)
0x21～0x3F ： 電源を入れた後、上記のファンクションを実行する
(0x20+0x01～0x1F)

戻り値 戻り値はありません。

プログラム例

```
#include        <iocslib.h>

main()
{
        int     code;   /* TVコントロールコード */
        int     i;
        int     j;

        code = 0x08;    /* テレビ/コンピュータ */
        TVCTRL(code);

        code = 0x0b;    /* チャンネルアップ */
        for (i = 1; i < 13; i++) {
                for (j = 0; j < 0x7fff; j++)
                        ;
                TVCTRL(code);
        }
        code = 0x1d;    /* コンピュータ画面 */
        TVCTRL(code);
}
```

TXBOX

レベル 0

書式 #include <iocslib. h>

```
void TXBOX (tboxptr) ;
strut TBOXPTR ＊tboxptr ;
struct TBOXPTR {
  unsigned short vram_page ; /＊  テキストのページ  ＊/
  short x ; /＊  始点 X 座標  ＊/
  short y ; /＊  始点 Y 座標  ＊/
  short x1 ; /＊  最終点までの X の長さ  ＊/
  short y1 ; /＊  最終点までの Y の長さ  ＊/
  unsigned short line_style ; /＊  ラインスタイル  ＊ /
} ;
```

機能 テキスト画面に四角形を描きます。

戻り値 戻り値はありません。

TXFILL

レベル 0

書　式

```
#include <iocslib.h>

void TXFILL (txfillptr) ;
struct TXFILLPTR *txfillptr;
struct TXFILLPTR  {
  unsigned short vram_page; /* テキストのページ */
  short x; /* 始点X座標 */
  short y; /* 始点Y座標 */
  short x1; /* 最終点までのXの長さ */
  short y1; /* 最終点までのYの長さ */
  unsigned short fill_patn; /* ぬりつぶしのパターン */
};
```

機　能　テキスト画面にぬりつぶした四角形を描きます。

戻り値　戻り値はありません。

TXRASCPY

レベル0

書　式

```
#include <iocslib.h>

void TXRASCPY (SOR_DEST, COPY, MODE) ;
int SOR_DEST ;
int COPY ; /*  コピーする個数  */
int MODE ; /*  コピーする画面  */
```

機　能

テキスト画面のラスタコピーを行います。
SOR_DEST には、ソース×256+デスティネーションを指定します。
各8ビットは4ラスタ単位の値となります。
COPY にはコピーする個数を指定します（4ラスタを1個とします）。
MODE にはコピーする画面と方向を指定します。
画面は同時に4面まで可能です。

ビット 0 ： テキスト0
ビット 1 ： テキスト1
ビット 2 ： テキスト2
ビット 3 ： テキスト3
ビット15 ： 0＝インクリメント
　　　　　　1＝デクリメント

戻り値

戻り値はありません。

TXREV

レベル0

書　式

```
#include <iocslib.h>

void TXREV (trevptr) ;
struct TREVPTR *trevptr;
struct TREVPTR {
  unsigned short vram_page; /* テキストのページ */
  short x; /* 始点のX座標 */
  short y; /* 始点のY座標 */
  short x1; /* 最終点までのXの長さ */
  short y1; /* 最終点までのYの長さ */
};
```

機　能　テキスト画面のリバースを行います。

戻り値　戻り値はありません。

TXXLINE

レベル 0

書式 #include <iocslib. h>

```
void TXXLINE (TXXPTR) ;
struct XLINEPTR ＊TXXPTR;
struct XLINEPTR {
  unsigned short vram_page; /＊ テキストのページ ＊/
  short x; /＊ 始点X座標 ＊/
  short y; /＊ 始点Y座標 ＊/
  short x1; /＊ 終点までのXの長さ ＊/
  unsigned short line_style; /＊ ラインスタイル ＊/
};
```

機能 テキスト画面に水平方向のラインを描きます。

戻り値 戻り値はありません。

TXYLINE

レベル 0

書　式

```
#include <iocslib.h>

void TXYLNE (TXYPTR) ;
struct YLINERTR *TXYPTR;
struct YLINERTR {
  unsigned short vram_page; /* テキストのページ */
  short x; /* 始点X座標 */
  short y; /* 始点Y座標 */
  short y1; /* 終点までのYの長さ */
  unsigned short line_style; /* ラインスタイル */
};
```

機　能　テキスト画面に垂直方向のラインを描きます。

戻り値　戻り値はありません。

UNLOCK

レベル 1

書　式　# include <doslib. h>

int UNLOCK (fileno, offset, len) ;
int fileno ; /* ファイルハンドル */
int offset ; /* ロック解除する部分へのオフセット */
int len ; /* ロック解除する部分の長さ */

機　能　ファイルのロック解除を行い、他のプロセスからのファイルアクセスを許可します。

fileno は、ロック解除したいファイルのファイルハンドルを指定します。

offset には、ファイル中のロック解除したい部分の、ファイルの先頭からのオフセットを指定します。

len には、ファイル中のロック解除したい部分のバイト数を指定します。

この UNLOCK 関数と LOCK 関数により、ファイルアクセスの排他制御を行い、自分以外のプロセスが勝手にファイルを書き変えるのを防ぐことができます。

戻り値　負の数ならエラーを表します。

参照関数　LOCK

プログラム例

```
#include        <stdio.h>
#include        <fcntl.h>
#include        <stat.h>
#include        <io.h>
#include        <doslib.h>

#define SIZE    256

static char     buf[SIZE];


main()
{
        int     fd;

        if ((fd = open("A:¥¥vjeb.dic", O_RDONLY|O_BINARY)) == -1)
                puts("can't open");
        else if (LOCK(fd, 0, SIZE) < 0)
                puts("can't lock");
        else if (read(fd, buf, SIZE) != SIZE)
                puts("can't read");
        if (UNLOCK(fd, 0, SIZE) < 0)
                puts("can't unlock");
        close(fd);
}
```

VDISPST

レベル0

書　式　# include <iocslib. h>

```
int VDISPST (ADDRESS, MODE, COUNTER) ;
unsigned char *ADDRESS ;  /*  割り込み処理アドレス  */
int MODE ;  /*  モード  */
int COUNTER ;  /*  カウンタ  */
```

機　能　V-DISPによる割り込みを、指定カウントごとに行います。

ADDRESSには割り込み処理アドレスを指定します（0ならば割り込み禁止)。

MODEには次の内容を指定します。

0：垂直帰線期間をカウント

1：垂直表示期間をカウント

COUNTERには0から255までの値を指定します。

戻り値　割り込み可能なときは0、すでに使用中の場合には0以外の値を返します。

VERIFY

レベル 1

書　式

include <doslib. h>

void VERIFY (flag) ;
int flag;　/＊　ベリファイフラグの設定モード　＊/

機　能

ベリファイフラグの設定を行います。
flag に設定できる内容は次の通りです。

0 ： ベリファイしない
1 ： ベリファイする

戻り値

戻り値はありません。

プログラム例

```
#include        <stdio.h>
#include        <doslib.h>

#define ON      1
#define OFF     0


main(int argc, char *argv[])
{
        if (argc == 1) {
                printf("verify は %s です。¥n",
                 (VERIFYG() ? "on" : "off"));
        }
        else if (strcmp(argv[1], "on") == 0) {
                VERIFY(ON);
                printf("verify を on にしました。¥n");
        }
        else if (strcmp(argv[1], "off") == 0) {
                VERIFY(OFF);
                printf("verify を off にしました。¥n");
        }
        else
                printf("Usage: verify [on|off]¥n");
}
```

VERIFYG

レベル 1

書　式　# include <doslib. h>

int VERIFYG ( ) ;

機　能　ベリファイフラグの設定状況を調べます。

戻り値　設定状況を示す次の値を返します。

0 : ベリファイは行わない
1 : ベリファイを行う

プログラム例

```
#include        <stdio.h>
#include        <doslib.h>

#define ON      1
#define OFF     0


main(int argc, char *argv[])
{
        if (argc == 1) {
                printf("verify は %s です。¥n",
                 (VERIFYG() ? "on" : "off"));
        }
        else if (strcmp(argv[1], "on") == 0) {
                VERIFY(ON);
                printf("verify を on にしました。¥n");
        }
        else if (strcmp(argv[1], "off") == 0) {
                VERIFY(OFF);
                printf("verify を off にしました。¥n");
        }
        else
                printf("Usage: verify [on|off]¥n");
}
```

VERNUM

レベル 1

書　　式　# include <doslib. h>

int VERNUM ( ) ;

機　　能　Human68k のバージョン番号を調べます。

戻 り 値　バージョン番号を返します。

バージョン番号の形式は次の通りです。

上位 16 ビット ＝ '68'
下位 16 ビット ＝ 整数部×256＋小数部

プログラム例

```
#include        <stdio.h>
#include        <doslib.h>

main()
{
        int     ver;            /* バージョン番号 */

        ver = VERNUM();
        printf("%c%c ", ver >> 24, ver >> 16);
        printf("version ");
        printf("%d.%d¥n", (ver >> 8) & 0xff, ver & 0xff);
}
```

VPAGE

レベル0

書　式

```
#include <iocslib. h>

int VPAGE (MODE) ;
int MODE;  /*  表示ページの ON/OFF 指定  */
```

機　能

グラフィック画面の表示ページを設定します。
MODE は表示ページの ON/OFF を指定します。

ビット0：0 = 0ページは表示しない
　　　　　1 = 0ページを表示する
ビット1：0 = 1ページは表示しない
　　　　　1 = 1ページを表示する
ビット2：0 = 2ページは表示しない
　　　　　1 = 2ページを表示する
ビット3：0 = 3ページは表示しない
　　　　　1 = 3ページを表示する

戻り値

終了コードを返します。

0：　正常終了
−1：　グラフィックは使用不可
−2：　ページ引数が指定どおりになっていない
−3：　指定されたページは現在のモードでは設定不可

WAIT

レベル 1

書　　式　# include <doslib. h>

int WAIT ( ) ;

機　　能　自分が直前に実行した子プロセスの終了コードを返します。

戻 り 値　終了コードを返します。

プログラム例

```
#include        <stdio.h>
#include        <doslib.h>

main()
{
        int     end_stas;                       /* 終了ステータス */

        end_stas = WAIT();

        exit(end_stas);
}
```

WINDOW

レベル 0

書　　式

```
#include <iocslib. h>

int WINDOW (SX, SY, EX, EY) ;
int SX;  /*  ウィンドウ左上のX座標  */
int SY;  /*  ウィンドウ左上のY座標  */
int EX;  /*  ウィンドウ右下のX座標  */
int EY;  /*  ウィンドウ右下のY座標  */
```

機　　能

グラフィック画面のウィンドウを設定します。
座標を指定するときは、SX≦EX、SY≦EY となっていなければなりません。
また、指定できるのはいずれも 0～1023 までです。

戻 り 値

終了コードを返します。

0 ： 正常終了
−1 ： グラフィックは使用不可
−2 ： 引数が指定通りになっていない

WIPE

レベル 0

書　　式　# include <iocslib. h>

```
int WIPE ( ) ;
```

機　　能　グラフィック画面をクリアします。

戻 り 値　終了コードを返します。

　0 ：　正常終了
−1 ：　グラフィックは使用不可

WRITE

レベル 1

書　　式

```
#include <doslib.h>

int WRITE (fileno, buffer, size) ;
int fileno ;  /*  該当ファイルのファイルハンドル  */
unsigned char *buffer ;  /*  書き出しデータ格納領域へのポインタ  */
int size ;  /*  書き出しバイト数  */
```

機　　能

bufferが指す領域からデータを、filenoで指定するファイルハンドルへ、sizeで指定するバイト数分、書き出します。

戻 り 値

書き出したバイト数を返します。もし、指定したsizeより小さい値や、0が返された場合はディスクフルが発生した事を表します。
したがって、ディスクフルでは負の数ではなく、正の数（0も）が返されることに注意して下さい。
負の数ならエラーを表します。

参照関数

READ

付　録

ユーザーライブラリの作り方

ソースファイルの作成
コンパイル
ライブラリファイル
ライブラリの外部宣言
リンク

C compiler PRO-68Kでは標準ライブラリ、IOCSコールライブラリ、DOSコールライブラリ等を提供していますが、これとは別にユーザー自身がオリジナルにライブラリを作成することができます。
本編では、ユーザーライブラリの作りかたを各ステップごとに述べるとともに、作成にあたっての注意事項についても述べていきます。

1　ソースファイルの作成

ライブラリもメインプログラムと同様に、エディタでソースファイルを作成します。
エディタの起動方法、および操作手順等については、「Human68k ユーザーズマニュアル」を参照してください。

2 コンパイル

ソースファイルの作成が終わったら、コンパイルを行います。コンパイルは、CCコマンドを起動することにより行います。
このとき注意しなければならないことは、CCコマンドを起動すると自動的にリンクまで行ってしまうので、リンク作業を抑止しなければならないということです。
というのは、ライブラリのリンクはメインプログラムとともに行う必要があるので、ライブラリのコンパイル段階でリンクを行うわけにはいかないからです。
そのようなわけで、ライブラリはオブジェクトコードだけ作成しておけば済みます（XC標準のライブラリもオブジェクトファイルとして提供されています）。
リンク処理の抑止は、Fcスイッチにより行います。ライブラリのコンパイルを行う場合の、CCコマンドのコマンドラインの具体例を以下に示します。

```
CC  /Fc  jstrlen.c⏎
```

なお、CCコマンド、およびCCコマンドのスイッチの詳細については、「Cユーザーズマニュアル」の「第2章　コンパイル」を参照してください。

3 ライブラリファイル

C 標準ライブラリの登録ファイルは clib. l です。
ユーザーが自身でライブラリを作成した場合、clib. l に登録することもできますし、またユーザー自身のファイルに登録することもできます。
また、ライブラリはオブジェクトファイル（拡張子 o）の形式でよいのですが、ファイルの管理上、ライブラリファイル（拡張子 l）に登録しておけば何かと便利です。
ライブラリファイルへの登録方法については、「アセンブラマニュアル」の「第 8 章　ライブラリアン」を参照してください。

4 ライブラリの外部宣言

新規に作成したライブラリをメインプログラムから呼び出す場合、ライブラリの外部宣言、および該当ライブラリで使用する（既存のインクルードファイルでは未定義の）記号定数や構造体などがある場合には、これらの定義が必要となります。
上記の外部宣言および定義を行う方法には、以下の２種類があります。

①新規にインクルードファイル（ユーザーインクルードファイル）を作成し、ここで外部宣言と定義を行う。

②メインプログラム中で外部宣言と定義を行う。

新規に作成したライブラリが汎用性に富んでおり、多数のプログラムで使用される可能性がある場合や、そのライブラリが使用する構造体や記号定数が数多い場合などでは、①の方法が有効でしょうし、そうでない場合には、②で済むこともあります。
これについては、ライブラリの作成者が適宜、選択してください。

5　リンク

ライブラリは、メインプログラムのリンク時にはじめて結合されます。
リンク時には、リンカのコマンドライン上にライブラリファイルの名前を指定します。
以下にコンパイラ、およびリンカのコマンドラインの具体例を示します。

```
cc jstrsample. c jstrlen. o ↵
lk jstrsample. o jstrlen. o clib. l ↵
```

上記の例では、ライブラリファイル名は、clib. l となっていますが、ユーザーがライブラリファイルを作成した場合には、代わりにそのファイル名を指定してください。

索引 50音順

50音順およびアルファベット順の索引は、すべて「C ライブラリマニュアル VOL.1」に対応しています。

ア

イ

エ

オ

カ

50音順

ス

セ

ソ

50音順

ロ

ワ

アルファベット順

50音順およびアルファベット順の索引は、すべて「Cライブラリマニュアル VOL.1」に対応しています。

関数索引 アルファベット順

※小文字関数は「C ライブラリマニュアル VOL.1」、大文字関数は「C ライブラリマニュアル VOL.2」のページ数を表記してあります。

A

B

アルファベット順

C

D

E

アルファベット順

F

G

H

I

アルファベット順

アルファベット順

アルファベット順

W

Z

機能別関数索引

※小文字関数は「C ライブラリマニュアル VOL.1」、大文字関数は「C ライブラリマニュアル VOL.2」のページ数を表記してあります。

バッファ操作

メモリ割りあて

文字の分類と変換

データ変換

itoa	整数を文字列に変換する	320
ltoa	long 型整数を文字列に変換する	340
strtod	文字列を倍精度の浮動小数点値に変換する	512
strtoul	X 進数の表す文字列を符号なし整数に変換する	517
strtol	数値文字列を long 型整数に変換する	515
uiota	符号なし整数を文字列に変換する	542
ultoa	符号なし long 型整数を文字列に変換する	543
uwtoa	符号なし short 型整数を文字列に変換する	550
wtoa	short 型整数文字列に変換する	550

ディレクトリ操作

関数名	機　能	ページ
chdir	ワーキングディレクトリを変更する	183
getcwd	ワーキングディレクトリを求める	284
mkdir	サブディレクトリを作成する	371
rmdir	サブディレクトリを削除する	438

ファイル操作

関数名	機　能	ページ
access	ファイルの許可設定を調べる	97
chgfa	ファイルの属性を変更する	184
chgft	ファイルの作成日時を変更する	185
chmod	ファイルの許可設定を変更する	189
chsize	ファイルの大きさを変更する	190
filelength	ファイルの長さを調べる	241
files	ファイルを検索しファイルの情報を求める	243
fstat	ファイルハンドルのステータス情報を求める	268
getfa	ファイルの属性を調べる	287
getft	ファイルの作成日時を調べる	289
isatty	文字デバイスかどうか調べる	317
mktemp	仮のファイル名を作成する	372
nfiles	次のファイルの候補を検索しファイルの情報を求める	393
rename	ファイル名を変更する	433
remove	ファイルを削除する	547
setmode	ファイルの変換モードを設定する	453

ストリーム入出力

低水準入出力

コンソール入出力

数値演算

プロセス制御

spawnl	引数リストによって子プロセスを実行する	463
spawnle	引数リストと環境によって子プロセスを実行する	463
spawnlp	引数リストと path 変数によって子プロセスを実行する	463
spawnv	引数配列によって子プロセスを実行する	463
spawnve	引数配列と環境によって子プロセスを実行する	463
spawnvp	引数配列と PATH 変数によって子プロセスを実行する	463
system	Human68k のコマンドを実行する	526
wait	直前に実行した子プロセスの終了コードを調べる	558

サーチとソート

関数名	機　能	ページ
bsearch	バイナリサーチを実行する	163
dqsort	double 型のデータをソートする	210
fqsort	float 型のデータをソートする	210
lqsort	long 型のデータをソートする	210
qsort	クイックソートを実行する	422
sqsort	short 型のデータをソートする	210
tqsort	ポインタ型のデータをソートする	210

文字列操作

関数名	機　能	ページ
strcat	文字列を追加する	487
strchr	指定文字の最初の位置を調べる	487
strcmp	2つの文字列を比較する	487
strcmpi	2つの文字列を比較する	487
strcpy	文字列をコピーする	487
strcspn	指定文字列の最初の位置を調べる	487
strdup	文字列をコピーする	487
stricmp	2つの文字列を英大/小文字を区別せずに比較する	487
strlen	文字列の長さを調べる	497
strlwr	すべての英大文字を英小文字に変換する	498
strncat	文字列を追加する	501
strncmp	2つの文字列を比較する	501
strncpy	文字列をコピーする	501
strnset	指定文字列に置き換える	501

strpbrk	指定文字列を探す	504
strrchr	指定文字の最後の位置を調べる	505
strrev	文字列を逆順にする	506
strset	指定文字列に置き換える	507
strspn	指定文字列以外の最初の位置を調べる	509
strstr	char 型引数 1 から char 型引数 2 を検索する	511
strtok	トークンを探す	514
strupr	すべての英小文字を英大文字に変換する	519
strins	文字列を挿入する	496
strmfe	ファイル名の拡張子を置き換える	499
strmfn	ファイル名を作成する	500
strmfp	ファイル名を作成する	500
stcgfe	ファイル名から拡張子をとり出す	484
strbpl	文字列からポインタ配列を作成する	486
stcgfn	ファイル名からノードをとり出す	484
strsrt	文字列を ASCII コード順にソートする	510

時　間

関数名	機　能	ページ
asctime	時間情報を文字列に変換する	108
ctime	時間を int 型整数値から文字列に変換する	207
clock	プロセスの使用時間を求める	194
difftime	時間の引算を行う	208
ftime	システム時間を調べる	271
gmtime	時間を整数値から構造体に変換する	296
localtime	時刻を地方時に変換する	334
mktime	ローカル時間をカレンダ時間に変換する	375
strftime	時間情報を書式にしたがって文字列に変換する	492
time	システム時間を調べる	532
tzset	環境時間変数から外部時間変数を設定する	540

日本語操作

関数名	機　能	ページ
iskana	カナ文字またはカナ句読点の判定を行う	318
iskpun	カナ句読点の判定を行う	318

BASIC 関連

IOCS コール

DOS コール

その他

シャープ株式会社
本　　　社　〒545 大阪市阿倍野区長池町22番22号
電子機器事業本部　〒329-21 栃木県矢板市早川町174番地
ＡＶＣシステム事業推進室
〒162 東京都新宿区市谷八幡町８番地　電話（03)3260-1161（大代表）
東京支社内　電子機器事業本部　ＡＶＣシステム事業推進室　ソフトウェア担当